(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 September 2001 (20.09.2001)

PCT

(10) International Publication Number
WO 01/69366 A2

(51) International Patent Classification⁷: G06F 3/00

(21) International Application Number: PCT/US01/07152

(22) International Filing Date: 7 March 2001 (07.03.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/524,011 13 March 2000 (13.03.2000) US

(71) Applicant: ERGODEX [US/US]; 1060 La Avenida, Mountain View, CA 94043-1422 (US).

(72) Inventor: RIX, Scott, M.; 505 Cypress Point Drive #104, Moutain View, CA 94043 (US).

(74) Agents: FRANKLIN, Eric, J. et al.; Swidler Berlin Shereff Friedman, LLP, Suite 300, 3000 K Street, NW, Washington, DC 20007 (US).

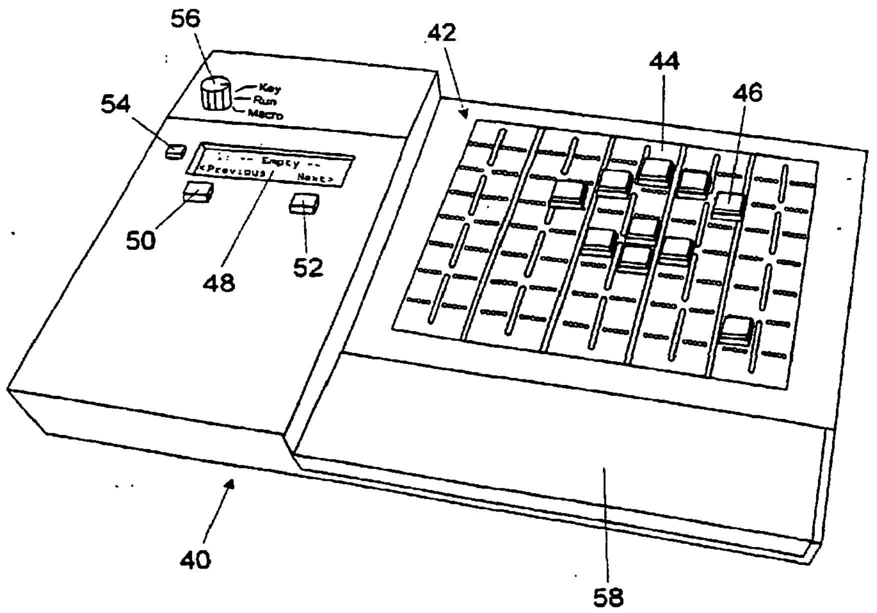
(81) Designated State *(national)*: JP.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: COMPUTER INPUT DEVICE WITH INDIVIDUALLY POSITIONABLE AND PROGRAMMABLE SWITCHES



(57) Abstract: A configurable computer input device. At least one switch is removably attached to a surface and is in communication with a processor. At least one function is assignable to acitvation of the switch. The at least one switch may be repositioned distances smaller than a length or width of the at least one switch. Circuitry is in communication with the at least one switch for assigning at least one function to activation of the switch. Circuitry communicates the at least one function to a host computer. Circuitry determines the actuation status of the at least one switch and communicates the actuation status to the host computer.

COMPUTER INPUT DEVICE WITH INDIVIDUALLY POSITIONABLE AND PROGRAMMABLE SWITCHES

Field of the Invention

The present invention relates to computer input devices. In particular, the present invention relates to an improved computer input device design that includes individually positionable and programmable switches. Also, the present invention relates to a system that includes the input device and to a method of programming a computer input device.

Background of the Invention

Traditional computer keyboard designs provide skilled typists with an effective instrument for data entry. For even a moderately trained user, the standard keyboard offers an adequate means to produce text or numerical data quickly and efficiently. However, the typical "QWERTY" style keyboard borrows much of its layout from the mechanical typewriters and adding machines that were originally invented at the end of the 19th century.

Practical computer use demands much more than the simple data entry tasks that these legacy layouts were created to perform. Many computer applications, including games, word processors, spread sheets, data bases, graphics programs, and computer aided design (CAD) applications, commonly rely on keyboard input to receive control instructions, initiate macros, or execute specific commands. Current keyboard layouts are not well suited to handle many of these functions easily, comfortably, or efficiently. Increasingly, these antiquated layouts are impelled to accommodate many more modern uses, exposing the inherent limitations of standard keyboard designs.

A good example of the shortcomings of standard computer keyboards is demonstrated by computer games. Many computer game players rely on a traditional computer keyboard as a primary input device. A typical game will assign one or more different keys on the standard keyboard to a corresponding action within the game. As computer games have become more involved and complex, more keys on the keyboard are required to control these virtual actions. This system often leaves game players using inconvenient or awkward key layouts that are not comfortable or effective. Furthermore, the often repetitive nature of key input required by video games amplifies the discomfort and awkward movements experienced from using key layouts not specifically adjusted for the unique hand anatomy of an individual game player.

Another failure of traditional keyboards stems from the inherent inability of the user to reposition individual key switches in a layout that reflects the associated key function. This shortcoming can also appear in video game controllers, particularly with children. Known available input devices, whether keyboards, game controllers, or others, offer very little alternatives to either the functional or physical layout of the input devices.

Programmable function keys are well known in the art. However, in general, once the overall location of keys is determined by a keyboard manufacturer, the user cannot easily change or modify the physical layout of a keyboard or game controller. Typically, the user must adapt to and memorize the locations of the keys in relation to their corresponding functions for each application. Often, the physical location of a key does not, in any way, indicate or allude to its underlying function. For example, a common action in computer graphics programs is to align two or more on-screen shapes. Typically, there is more than one geometrical way to align these shapes, including by their top, middle, bottom, left, center, or right, as well as by combinations of these, such as aligning the shapes by their top-left corners. Although it is possible to use traditional programmable function keys to perform these actions, it is generally not possible to reposition the keys in a way that intuitively reflects their particular function.

Summary of the Invention

The present invention provides a configurable computer input device. The device includes at least one switch removably attachable to a surface and in communication with a processor.

At least one function is assignable to activation of the switch. The at least one switch may be repositioned at distances smaller than a length or width of the at least one switch. The device also includes circuitry in communication with the at least one switch for assigning at least one function to activation of the switch. Circuitry communicates the at least one function to a host computer. The device additionally includes circuitry for determining the actuation status of the at least one switch and communicating the actuation status to the processor.

Also, the present invention concerns a configurable computer input device. The device includes at least one switch removably attachable to a receiving surface. The at least one receiving surface receives the at least one switch at a plurality of positions. A distance separating any two positions may be smaller than a length or a width of the at least one switch. A memory stores data comprising a mapping of at least one electronic, alphanumeric, or keyboard function to the at least one switch when the at least one switch is removably attached to a position of the receiving surface. The device also includes circuitry for scanning the at least one switch when the at least one switch is removably attached to a position of the receiving surface and for determining the actuation status of the at least one switch. A controller is responsive to the circuitry for scanning and for consulting the memory to obtain the at least one function mapped to the at least one switch upon actuation of the at least one switch. Circuitry transfers the at least one function obtained by the controller to a host computer with which the device communicates.

Additionally, the present invention relates a computer system that includes a configurable computer input device. The input device includes at least one switch removably attachable to a surface and in communication with a processor. The surface includes at least one matrix of receptacles tnat the at least one switch is removably attachable to at a plurality of positions. The at least one switch may be repositioned on the matrix of receptacles at distances smaller than a length or width of the at least one switch. At least one function including at least one electronic, alphanumeric or keyboard function is assignable to activation of the switch. Circuitry communicates with the at least one switch for assigning at least one function to activation of the switch. Circuitry communicates the at least one function to a host computer. A memory stores data including a mapping of the position of the at least one switch in the at least one matrix of receptacles and the at least one function assigned to the at least one switch. Circuitry determines the actuation status of the at least one switch and communicates the actuation status to the processor. A host computer includes a microcomputer or a video game computer.

Furthermore, the present invention concerns a method for generating input to a computer. A configurable computer input device is provided including at least one switch removably attachable to a surface and in communication with a processor. At least one function is assignable to activation of the switch. The at least one switch may be repositioned on the surface at distances smaller than a length or width of the at least one switch. Circuitry communicates with the at least one switch for assigning at least one function to activation of the switch. Circuitry communicates the at least one function to a host computer. Circuitry determines the actuation status of the at least one switch and communicating the actuation status to the processor. At least one input is provided from an existing computer input device. At least one input is received from the existing computer input device. The at least one input from the existing computer input device is assigned to the at least one switch.

Still other objects and advantages of the present invention will become readily apparent by those skilled in the art from the following detailed description, wherein it is shown and described only the preferred embodiments of the invention, simply by way of illustration of the best mode contemplated of carrying out the invention. As will be realized, the invention is capable of other and different embodiments, and its several details are capable of modifications in various obvious respects, without departing from the invention. Accordingly, the drawings and description are to be regarded as illustrative in nature and not as restrictive.

Brief Description of the Drawings

The objects and advantages of the present invention will be more clearly understood when considered in conjunction with the accompanying drawings, in which:

Fig. 1 represents a perspective view of an embodiment of a programmable computer input device according to the present invention;

Fig. 2 represents a rear view illustrating connection and control locations that may be included in the embodiment of the invention shown in Fig. 1;

Fig. 3 represents an embodiment of a system according to the present invention;

Fig. 4 represents a block diagram that illustrates functional components that may be included in an embodiment of a programmable computer input device according to the present invention;

Fig. 5 represents an overhead view of an embodiment of an attachment surface according to the present invention that includes a matrix of connections;

Fig. 6 represent a close-up view of a portion of the embodiment of the attachment surface illustrated in Fig. 5, showing an embodiment of electrical connections that may be included in a matrix of connections;

Fig. 7 represents a table showing an embodiment of a distribution of matrix scanner row and column connections for a set of matrix pads accord according to the present invention;

Fig. 8A represents an exploded side view of an embodiment of key switch module components according to the present invention;

Fig. 8B represents a side view of an embodiment of an assembled key switch module according to the present invention;

Fig. 8C represents an exploded top view of an embodiment of key switch module components according to the present invention;

Fig. 9 represents a top view of three key switch modules according to an embodiment of the present invention in relation to an embodiment of an attachment surface that includes a matrix of receptacles, illustrating some different ways individual key switch modules may be positioned into the matrix;

Fig. 10 represents a block diagram illustrating an example of input and output details of an embodiment of a microcontroller development board that may be utilized according to the present invention;

Fig. 11 represents an electrical schematic diagram illustrating an embodiment of input/output (I/O) circuits for an embodiment of a liquid crystal display and for an embodiment of a mode selection and button control circuit according to the present invention;

Fig. 12 represents an electrical schematic diagram showing an embodiment of I/O circuits among a host computer, a matrix scanner, a standard keyboard, and a microcontroller; and a system reset circuit according to the present invention;

Fig. 13 represents a block diagram illustrating an embodiment of a data memory usage map according to the present invention;

Fig. 14 represents a block diagram illustrating an embodiment of a code memory usage map according to the present invention;

Fig. 15 represents a flowchart outlining an embodiment of a program startup process according to the present invention;

Fig. 16 represents a flowchart outlining an embodiment of program interrupt service routines according to the present invention;

Fig. 17 represents a flowchart outlining an embodiment of a program run mode routine according to the present invention;

Fig. 18 represents a flowchart outlining an embodiment of a program key mode routine according to the present invention;

Fig. 19 represents a flowchart outlining an embodiment of a program macro mode routine according to the present invention;

Fig. 20A represents a close-up top view a portion of an input device according to the present invention, illustrating an embodiment of a display, three control buttons, and a mode selection switch as they could appear during a run mode according to the present invention;

Fig. 20B represents a close-up top view of the portion of an input device shown in Fig. 20A, illustrating an embodiment of a display, three control buttons, and a mode selection switch as they could appear during a key mode according to the present invention;

Fig. 20B represents a close-up top view of the portion of an input device shown in Fig. 20A, illustrating an embodiment of a display, three control buttons, and a mode selection switch as they could appear during a macro mode according to the present invention;

Fig. 20D represents a close-up top view of the portion of an input device shown in Fig. 20A, illustrating an embodiment of a display, three control buttons, and a mode selection switch as they could appear during a clear label mode according to the present invention;

Fig. 21 represents a perspective view of an embodiment of the present invention that includes a non-contiguous, curved, two-handed attachment surface that includes a matrix of receptacles arranged in two separate regions according to the present invention;

Fig. 22 represents a top view of another embodiment of the present invention that includes a matrix of receptacles including input function posts and external connection wires;

Fig. 23 represents a top view of an embodiment of the present invention including matrix of receptacles housed in an embodiment of a keyboard housing according to the present invention;

Fig. 24 represents a top view of an embodiment of the present invention including a matrix of receptacles housed in an embodiment of a game controller according to the present invention;

Fig. 25 represents a schematic drawing of an embodiment of a device bypass circuit that permits keyboard traffic to pass to a host computer when an input device according to the present invention has power turned off or is not connected to a power supply;

Fig. 26 represents a perspective view of an embodiment of an input device according to the present invention that includes an embodiment of an overlay card mounted on the device; and

Fig. 27 represents a perspective view of an embodiment of a system according to the present invention that includes wireless communication between elements of the system.

Detailed Description of the Invention

The present invention provides a physically configurable and programmable and reprogrammable computer input device. As such, the present invention provides advantages that are unknown in computer input devices. Along these lines, traditional computer keyboards provide rigid, static, standardized, and generally inflexible devices. Known keyboards demand that users adapt to a single particular configuration or a limited number of specific alternate configurations. The same may be said of other computer input devices, such as game controllers, mice trackballs, and others.

The present invention overcomes these and other shortcomings of known computer input devices. Along these lines, the present invention has a number of objects and advantages. Every embodiment of the present invention need not exhibit each advantage or object. For that matter, is not necessary that an embodiment exhibit any of the object or advantage discussed below.

One advantage of the present invention it that it provides a computer input device that permits a user to specify the location of individual key switches and to change quickly and easily the location when desired. Another advantage of the present invention is that it provides a computer input device that permits fine adjustment of individual key switch locations, such that changes in location can be made that are smaller than the width or height of key switch. Along these lines, the keys, buttons, or other actuated members of a devices according to the present invention may be moved in non-whole number multiples of dimensions of the actuated members. The actuated members may also be located relative to each other at other than whole number multiples of dimensions of the actuated members.

Additionally, advantages of a computer input device according to the present invention can include that the device can provide for assignment or reassignment of each individual key switch function without affecting the assigned function of other key switches. Furthermore, a computer input device according to the present invention can permit a user to program any individual key switch to emulate at least one actuation of a key of a keyboard, including a macro of multiple key actuations. Input from other input devices other than a keyboard may also or alternatively be included in the functions of a key switch according to the present invention. Along these lines, an input device according to the present invention can provide a computer input device that allows users to program and store the function of separate key data sets and to allow the users to choose among these data sets, so that the users may employ the same or different key layouts with separate computer applications. Again, input from other input devices may alternatively or additionally be included in the data sets.

Still further, advantages of an input device according to the present invention can include providing a computer input device that does not require special software or device drivers to be installed on the host computer with which it is used. Also, the present invention can provide a computer input device that may operate in conjunction with a standard computer keyboard or other standard input devices, such that a user may continue to use these standard devices in a normal manner.

While there has been some attempt in the prior art to produce more dynamic keyboard layouts, such attempts fall far short of the input device according to the present invention. Along these lines, entire blocks of keys are exchanged for other blocks or with other devices, such as a trackball. Such devices do not permit rearrangement of the location or function of individual keys within the functional blocks of keys.

Other solutions have included keyboards in which one modular key may be exchanged with another. This replaceable key module design facilitates the exchange of a broken key with an operational one. However, the overall layout of the keys within the keyboard, and their relative positions, remains fixed. Each key position is predetermined by the manufacturer of the keyboard and there is a one-to-one relationship between each key module and its corresponding panel opening. In other words, relocation of the keys is not permitted.

Even devices that permit altering the function of a keyboard key include fixed key switches mounted in a keyboard matrix arrangement and do not permit the physical relocation or programmability of the input device according to the present invention. Along these lines, the design may require including a key switch at every potential location, thereby increasing the manufacturing complexity and cost of the keyboard. Also, the position of each key is limited to immovable switch positions determined by the manufacturer. When positioning individual key caps, the user merely has the option either to use a particular fixed switch, or to leave it empty. Such designs do not easily support fine adjustments to key position, characterized by changes smaller in scale than the width or height of a single key cap.

In a broad sense, the present invention includes a configurable computer input device. The device includes at least one switch removably attachable to a surface. The switch may be housed in a structure in the form of a key of a keyboard. Alternatively, the switch could be housed in a structure such as a button from a game controller. Such a button could have the shape of any game controller button. In reality, a switch according to the present invention could be housed in any structure. Along these lines, other forms that a switch may have include a button switch, a swiveling key-cap top, a tilting key-cap top, a swiveling and tilting key-cap top, and keyboard keys having non-standard shapes and sizes, such as in the shape of an arrow. Key cap labeling may also be utilized, applied directly to a key or inserted into a label sleeve on a key cap. Any other form may also be utilized.

A switch according to the present invention may be removably attached to a surface, referred to below as the attachment surface. The attachment surface can include any surface. For example, the present invention could include a switch, attached to virtually any surface. Examples of a surface include at least one of a mouse, a monitor, a keyboard, a desk, a work surface, a keyboard tray, a switch tray, a switch platform, a chair, a computer, a printer, and/or any other surface. Such a switch could be in wired or wireless communication with a processor and/or other circuitry.

A wireless key switch would not need to be plugged in to a typical matrix. Along these lines, wireless key switch modules use known broadcast techniques, such as radio-frequency or infra-red transmission, to indicate their activation status to a receiving unit. By including an attachable backing, such as an adhesive, suction, or magnetic surface, the wireless key switch modules allow the user to removably position individual keys in many useful and novel locations such as those described above. Any wired communication described herein could also be replaced with wireless communication means.

Other attachment surfaces may also be included in an input device according to the present invention. Along these lines, the attachment surface can include a matrix of receptacles. The matrix of receptacles may have a variety of shapes and sizes and may be housed in a variety of housings. For example, the matrix of receptacles may have a shape and be provided in a housing such as that illustrated in Fig. 1. Alternatively, the matrix of receptacles could have a size and be housed in a housing such as that illustrated in Fig. 23, where the matrix of receptacles has a size that permits it to be housed with a standard keyboard. The matrix of receptacles could also have a shape and size that permits it to be used with a video/computer game controller such as the embodiment illustrated in Fig. 24.

The present invention may be partially or fully incorporated into another computer input device. According to one embodiment, the device is incorporated with a standard keyboard, as shown in Fig. 23. As shown in Fig. 23, the matrix 42 may be included in the same housing as the standard key layout. Advantages of such an embodiment include the ability to use a single matrix scanner for both the standard keys and the added matrix locations. The combined design may also eliminate the need for external pass-through connections since the device may connect to the host computer 70 in the same manner as a standard keyboard.

In a further modification of this embodiment, one or more matrix pads 106 may share the row/column designation of keys on the standard keyboard 72. The functions of the matrix 42 positions may be assigned by the host computer, as described above, or an indicator could be used to differentiate signals sent by the matrix 42 from signals sent by the standard keyboard 72 keys.

As is apparent from the above discussion, the matrix of receptacles may be contiguous or non-contiguous. The matrix of receptacles may be planar. Alternatively or additionally, at least a portion of the matrix of receptacles may be non-planar.

Regardless of the shape and size of the matrix of receptacles, the matrix may receive pins extending from at least one switch. The pins may be attached to and electrically connected to the at least one switch. Significantly, as described in greater detail below, the receptacles and pins can permit the switches to be positioned at intervals smaller than the dimensions of the structures that house the switches. Along these lines, housing of the switches may be repositioned at distances other than whole number multiples of the dimensions of the switches. Typically, the dimensions of the housing of the switches are referred to herein as the dimensions of the switches.

Furthermore, if an input device according to the present invention includes more than one switch, the switches may be arranged on the matrix of receptacles such that they are not aligned. However, switches may be aligned if desired. In some embodiments, some switches may be aligned while others are not. This is one of the great advantages of the present invention; the switches may be arranged as desired by a user.

At least one function is assignable and/or reassignable to activation of the at least one switch included in an input device according to the present invention. The at least one function can include at least one input provided to a computer. Along these lines, the at least one function can include any electronic, alphanumeric, or keyboard function. Along these lines, the at least one function can include at least one function including movement of a joystick, actuation of a mouse button, actuation of a button or other member on a game controller, and/or actuation of a keyboard key. The at least one function could also include any other input from any other computer input device.

With respect to use of the present invention with other computer input devices including joysticks and game controllers, Fig. 24 illustrates an example of another input device that the present invention may be utilized with. Many common game controllers, including those made by Sega, Nintendo, Sony, and Atari, provide fixed switch positions that frequently are not comfortable to use, especially for children who typically have smaller hands than adults. Providing a matrix 42 with various game controller switches 212, and using the standard game controller interfaces, the present invention can allow a user to define their own game controller layout. This design may be utilized with analog and digital controller input functions.

Not only can the function include any one or more of the above functions, it can also include timing, cadence, and sequence of functions. Along these lines, function could include how hard a keyboard key is struck, the length of time the key is depressed, and the timing to the next stroke. This type of timed macro could reproduce both a key sequence and timing between key presses. Such a macro could be useful for game applications where key input timing is important. The direction that a joystick is moved and other variables could also be represented by the at least one function. In a computer game scenario, pushing one button on a device according to the present invention could result in a spaceship turning with a certain speed while moving in a certain direction and firing a certain gun with a selected rapidity, among other functions. A device according to the present invention could be modified to operate for chord key input. In such a configuration, input functions may be programmed to simultaneous combinations of key presses and/or other inputs instead of just single key presses or other input. As apparent from the above discussion, the possibilities for functions of the present invention are endless.

A configurable computer input device according to the present invention also includes circuitry in communication with the at least one switch for assigning at least one function to activation of the switch. The at least one function is discussed above in greater detail. The nature of the circuitry is discussed below in greater detail. Although, as discussed above, the communication between the function assigning circuitry and the at least one switch may be wired or wireless.

Additionally, an input device according to the present invention may include circuitry for communicating the at least one function to a host computer. As with the communication between the function assigning circuitry and the at least one switch, the communication between the host computer and the circuitry for communication the at least one function to the host computer may be wired or wireless. While a "host computer" is referred to herein, it is not necessary that the host computer actually be a traditional microcomputer. The host computer could be a video game console, or any other device.

In addition to the above circuitry, an input device according to the present invention may also include circuitry for determining the actuation status of the at least one switch and communicating the actuation status to a processor. As with the communication between the function assigning circuitry and the at least one switch, the communication between the processor and the circuitry for communicating the actuation status of the at least one switch to the processor may be wired or wireless. The actuation status determining circuitry may scan the at least one switch to determine the status of the switch. Functioning of this and other circuitry is discussed below in greater detail.

An input device may also include memory for data storage. The data can include a mapping of the position of the at least one switch in the at least one matrix of receptacles and the at least one function assigned to the at least one switch. An input device according to the present invention where the attachment surface includes a matrix of receptacles typically includes a memory. The mapping of functions may also reside in the memory of the host computer.

To facilitate control of operation of an input device according to the present invention may include a controller. Among the functions that a controller may carry out are receiving the actuation status of the at least one switch, determining the function assigned to activation of the switch, and transmitting the at least one function to the circuitry for communicating the at least one function to the host computer. A controller may also carry out any other desired function.

In determining the function assigned or mapped to the at least one switch, the controller may consult a memory such as the memory described above. The controller may then transfer the function to the circuitry for communicating the function to the host computer.

A significant advantage of an input device according to the present invention is that the at least one switch of the input device may be operated nearly simultaneously with other input devices, such as keyboards, mice, and trackballs, in communication with the host computer. Many computer input devices do not operate in such a manner. Rather, generating input on an existing computer input device prevents the possibility of generating input on another input device. The computer will not receive the additional input.

The present invention also includes a method for generating input to a computer. The method includes providing at least one input from a computer input device other than the input device of the present invention. As described above, examples of such input devices include keyboards, mice, joysticks, and game controllers, among others. The at least one input is recorded.

Then, the at least one input is assigned to the at least one switch of an input device according to the present invention, wherein actuation of the at least one switch results in the at least one input. The at least one switch is then actuated, providing the at least one input to the computer.

As described above, an input device according to the present invention permits keys, or switches more generically, to be arranged in any desired arrangement. In many cases, the arrangement of keys on a keyboard, game controller, or other input device is not the most desirable for a particular user and/or carrying out a particular function. One example of a desired function-key layout for aligning geometrical shapes in a graphics program is shown as follows:

Top-Left	Top	Top-Right
	Center	
Left	Center-Middle	Right
	Middle	
Bottom-Left	Bottom	Bottom-Right

The positioning shown above is unavailable with the traditional function key group that is fixed across the top or to the left of a standard alphanumeric keypad.

The present invention solves the above and other problems by providing a computer input device that permits a user to position individual key switches in a custom manner and to program the individual function of those key switches. The key switch layout and function can be changed easily to suit the needs of the user. Typically, the invention uses standard interface protocols, thereby eliminating the need to employ special device driver(s) or interpretation software on a host computer. Furthermore, the present invention permits standard input devices, such as keyboards, to be used concurrently and without noticeable interference.

The present invention will now be explained in greater detail with respect to one particular embodiment and some alternate embodiments. These embodiments and the discussion are illustrative of the present invention and should by no means be interpreted as the only embodiments. The principles described below can apply to other embodiments. Also, alternative means for accomplishing structures and functions described below are possible. Those of ordinary skill in the art would be able to make substitutions and/or deletions without undue experimentation.

Fig. 1 offers a perspective view of one embodiment of the present invention. The embodiment illustrated in Fig. 1 includes a housing 40. This embodiment includes an attachment surface that includes a matrix of receptacles.

The matrix of receptacles is contained within the housing. Along these lines, the matrix of receptacles in the embodiment shown in Fig. 1 is arranged at top of the housing 40. This embodiment of the matrix of receptacles includes five solderless breadboard terminal strips, or terminal strips 44. Such strips are available from Digi-Key Corporation of Thief River Falls, MN, among other sources.

The terminal strips 44 may be connected to the housing 40 in a rectangular block, to form the matrix of receptacles or key-switch plug-in matrix, or matrix 42. The matrix 42 provides a plug-in surface to receive a plurality of modular positionable key switch assemblies, or key switch modules 46. A user may reposition the key switch modules 46 within the matrix 42. The terminal strips 44 described here are modified for their use in the present invention. One example of a detailed description of the matrix design and construction is provided below.

To facilitate use of the present invention, an overlay card 218 may be used to indicate the programmed functions of the key switch modules 46. Fig. 26 illustrates an embodiment of an overlay card. The overlay card 218 may include well-known functional indicators such as symbols 220, functional labels 222, and configuration identification labels 224. Separate overlay cards 218 may be applied for different data set configurations of the device. An overlay card could be utilized with any embodiment of the present invention. For example, an overlay card could be utilized with a game controller or the embodiment illustrated in Fig. 23, or any other embodiment.

The present invention may also include a display. The display can assist in the operation of the device. Along these lines, the display can provide a readout of keystrokes that comprise a function assigned to actuation of a switch of the input device.

In the embodiment illustrated in Fig. 1, the display is arranged in the housing 40 along with the matrix 42. This embodiment of a display includes a liquid crystal display, or LCD 48. Any other display may also be utilized, if the device includes a display. The embodiment of the display 48 shown in Fig. 1 displays device status and programming information to the user. The display may also display other function(s).

As described above, the present invention also includes a method for generating input to a host computer. For controlling the operation of an input device, including programming and subsequent provision of input to a host computer, an input device according to the present invention may include at least one control element. The at least one control element can determine whether the input device is "learning" the at least one function that is to be assigned to the at least one switch, operating to permit actuation of the at least one switch to carry out the at least one function, or perform other function(s).

The embodiment shown in Fig. 1 includes a plurality of controls. Along these lines, the embodiment shown in Fig. 1 includes a left selection button, or left button 50; a right selection button, or right button 52; a label button 54; and a rotary mode selection switch, or mode switch 56. The left button 50 and right button 52 can permit a user to select a data set of the device.

The present invention may also include a label button 54, which allows the user to input the displayed name of the selected device data set. A mode switch may be included to permit a user to choose an operation mode of the device. Possible operational modes can include programming modes and operational modes. One program mode permits programming a single key to a switch position. Another programming mode could permit programming a macro, or plurality of functions, to a switch position. Another example of a mode includes a run or functional mode, wherein actuation of the at least one switch results in the desired input to a host computer.

The housing 40 may also include a wrist rest 58 to make using the device more comfortable.

Fig. 2 illustrates a rear view of the embodiment shown in Fig. 1. As illustrated in Fig. 2, an input device according to the present invention may include at least one connector for making wired connections between a device according to the invention and a host computer or other device. Whether or not an input device according to the present invention includes a connector, the connections between the device and a host computer or other device may be wireless.

The embodiment shown in Fig. 2 includes a plurality of connectors. These connectors include a power connector 62, a host computer connector or host connector 64, and a keyboard connector 66. The power connector 62 in the embodiment shown in Fig. 2 is a male center-post connector designed to receive a 5-Volt, 0.8 Amp, direct-current power source to power the device. Although with certain design changes it is possible to use power provided directly by the host computer, as with most standard keyboards, an external power source is included in this embodiment to provide a more generous power budget.

Additional operational controls may also be included on the rear of a device according to the present invention. Along these lines, as shown in Fig. 2, a device according to the present invention may include a clear data set or clear button 68 that permits a user to erase all programmed switch positions within a selected data set. The device may also include a power switch 60. The power switch 60 may be used to apply or remove electrical power to the device, turning the device on or off. It is not necessary that such operational controls are located on the rear of the device; they could be provided on any other surface. The controls could also take different forms, such as touch pads.

It is not even necessary that the device include such controls. Along these lines, clearing a function could take place automatically when a new function is entered. Also, if the device were powered directly by a host computer, in a manner similar to a standard computer keyboard, then the power switch would be superfluous.

To permit a device to be in wired connection with a host computer, keyboard, and/or other device(s), the device according to the present invention could include at least one connector. Along these lines, the rear of the device may include such connectors. For example, the embodiment shown in Fig. 2 includes a connector 64 for connecting the device to a host computer and connector 66 for connecting a keyboard to the device. The device according to the present invention may include more or less connectors.

Any type of connectors may be utilized according to the present invention. For example, host connector 64 and keyboard connector 66 may be PS/2 style female connectors. Such connectors are typically known in the art as six-position miniature-DIN connectors. One source of such connectors is Digi-Key.

In the embodiment of the present invention shown in Fig. 3, the device is connected between a host computer 70 and a standard keyboard 72. A typical keyboard connection cable 74, which usually connects directly to a PS/2 keyboard port (not shown) of the host computer 70, instead connects to the keyboard connector 66 of the device according to the present invention. A host connection cable 76 links the device from the host connector 64 to the keyboard port of the host computer 70. As described in this basic embodiment, the device can act as a wedge between the standard keyboard 72 and the host computer 70. The device can relay both the data signals generated by the attached standard keyboard 72 and its own data signals directly to the keyboard port of the host computer 70. However, other arrangements and data transmission paths are also possible.

The device according to the present invention can be compatible with computers and keyboards operating under the widely used PS/2, or PC AT, standard keyboard communication protocol, originally defined by International Business Machines Corporation of Armonk, NY. However, one of ordinary skill in the art could modify the present invention without undue experimentation to use any other standard or custom keyboard communication protocol. Examples of other protocols include IBM PC XT standard, Apple Desktop Bus keyboard interface, ASCII parallel keyboard interface, standard serial port keyboard interface, and Universal Serial Bus (USB) keyboard interface.

The block diagram in Fig. 4 shows one possible relationship among major functional components of an embodiment of a system according to the present invention. This embodiment includes an attachment surface that includes a matrix of receptacles described above. The matrix, such as matrix 42 shown in Fig. 1, may be scanned by a matrix scanner 78 to determine the location of any pressed key switch modules 46. The matrix scanner represents an example of an embodiment of circuitry for determining the actuation status of the at least one switch.

The position of any activated key switch modules 46 detected by the matrix scanner 78 may be sent to a microcontroller development board, or microcontroller 80 through an input/output circuit A 82. Data may be transferred between the standard keyboard 72 and the microcontroller 80 through an input/output circuit B 84. Also, data may be transferred between the host computer 70 and the microcontroller 80 through an input/output circuit C 86. A memory 88 may be divided into a code space and a data space to provide for both the control software and storage requirements of the microcontroller 80.

An LCD circuit 90 can permit information from the microcontroller 80 to be displayed on a display, which in this case is an LCD 48 shown in Fig. 1. A mode selection and button control circuit 92, which includes the left button 50, the right button 52, the label button 54, the mode switch 56, and the clear button 68, provides means for the user to input commands to the microcontroller 80. Finally, a system-reset circuit 94 provides means to initialize the device status.

Fig. 5 provides a more detailed image of the matrix of receptacles 42 illustrated in Fig. 1. As stated earlier, the matrix 42 may include five vertical terminal strips 44 arranged in a rectangular block. Each terminal strip 44 may be separated from an adjacent strip with a spacer 104. Solderless terminal strips are well known in the art. Such strips are customarily used for the design, construction, and testing of prototype electrical circuits by providing a means to connect electrical components temporarily.

Integrated-circuit pins, wires, or other electrical components connected to the at least one switch may be inserted into a connection hole of a terminal strip. A metallic spring clip at the base of the hole can be provided to form both an electrical and a mechanical connection to the component(s) through contact friction. A standard, unmodified terminal strip typically includes sixty-four rows of ten solderless connection tie points, or tie points, 96 split evenly into two columns by a central divider 98.

The terminal strips 44 that may be included in a device according to the present invention can be modified to include framing rows 100. The framing rows 100 may be formed by inserting tie-point plugs 102 into the tie points 96 of every eighth row of the matrix 42. The tie-point plugs 102 may be utilized to inhibit use of tie points 96 in a framing row 100 when selecting a position for a key switch module 46 in the matrix 42. The tie point plugs 102 may be manufactured by cutting the round, flat ends of common steel fabric pins to a length such that they may be inserted into the tie points 96 of framing rows 100, effectively blocking their use. The tie-point plugs 102 typically lie flush with the surface of the matrix 42.

As shown in Fig. 5, framing rows 100, central dividers 98, spacers 104, and edges of the matrix 42 may combine to form an array of matrix pads, or pads, 106. Each pad 106 may include a grid of thirty-five tie-points 96 arranged in seven rows and five columns. The pads 106 may be the useable portions of the matrix 42 into which the key switch modules 46 may be inserted.

Many different matrix designs may be substituted for the matrix 42 shown in Fig. 5. One embodiment may utilize a matrix 42 that maps each tie-point 96 separately to a matrix scanner 78, eliminating the pads 106, and a corresponding need for framing rows 100, spacers 104, and central dividers 98 described in the basic embodiment. This embodiment provides a slightly greater freedom in key switch module 46 placement, at the expense of an increase in the number of tie points 96 monitored.

According to another embodiment, the matrix 42 design may be modified from the planar, rectilinear structure described in the basic embodiment. The matrix 42 design ma ncorporate any reasonable morphology or topology. Further re, the matrix 42 design does not need to be contiguous.

Along these lines, two or more separate regions of the matrix may be used to supply convenient areas where key switch modules may be inserted and monitored. For example, Fig. 21 illustrates a "two-handed" matrix 42 design with separate matrix regions 206A and 206B, one for a left hand and one for a right hand. The matrix regions 206A, 206B have curved surfaces to provide the user with a wide choice in selecting comfortable hand positions. Key switch modules 46 may be positioned anywhere within these regions.

Fig. 6 represents a close-up view of a portion of the matrix of receptacles circled in Fig. 5. As shown in Fig. 6, the terminal strips 44 may be modified such that alternating rows within the pad 106 are connected electrically. A matrix scanner row data line, or row line, 108 may connect the first, third, fifth, and seventh rows of tie points 96 within the pad 106. A matrix scanner column data line, or column line, 110 may connect the second, fourth, and sixth rows of tie points 96 within the pad 106. Neither the row line 108 nor the column line 110 are typically visible from the top of the matrix pad. The line schematics added to Fig. 6 are for illustrative purposes. The electrical connections for the row line 108 and the column line

110 to the tie points 96 may be created by first removing the adhesive backing from the bottom of the terminal strips 44 to expose the back of the embedded metallic spring clips. Jumper wires may then be soldered directly to the tie-point spring clips, electrically joining the alternating rows as shown in Fig. 6. Of course, any other means that provides the desired connections may also be utilized.

The row line 108 from each pad 106 may be connected to a row input of the matrix scanner 78. The column line 110 from each pad 106 may be connected to a column input of matrix scanner 78. The matrix scanner 78 used for this embodiment may be a keyboard controller circuit. Such a circuit could be obtained from an existing keyboard. One example of such a keyboard is a Chiconey Pro Keyboard, Part #70082, available from a retail outlet of CompUSA, Inc. of Dallas Texas.

The matrix scanner 78, like many generic keyboard controllers, typically accepts "row" and "column" inputs to identify a key press from a standard keyboard. Each key in a standard keyboard may be assigned both a row and a column. Pressing or actuating a key connects the assigned row and column lines, uniquely identifying the key. Although multiple keys may be assigned the same row line or the same column line, each key has a unique row-column combination.

The matrix scanner 78 utilized in this embodiment of the present invention contains eighteen column inputs, 0 through 17, and eight row inputs, 0 through 7. These input lines may be reassigned to scan the matrix 42 as shown in the table in Fig. 7. Each box in the table may correspond to the position of a pad 106 in the matrix 42.

The boxes in the table have four lines that describe an original key assignment 112, a hexadecimal byte code 114, a row input number 116, and a column input number 118. The original key assignment 112 lists the name of the standard key that originally was associated with the given combination of row input number 116 and column input number 118 prior to salvaging the matrix scanner 78 for this embodiment. The hexadecimal byte code 114 lists the byte code generated by the matrix scanner 78 for the given combination of row input number 116 and column input number 118. As shown by the table, each pad 106 in the matrix 42 has a unique row-column designation and, therefore, a unique hexadecimal byte code 114 representation from the matrix scanner 78. This unique hexadecimal byte code 114 may be used by the microcontroller 80 to determine which pad 106 was activated by a key press from an inserted key switch module 46.

Fig. 8A, Fig. 8B, and Fig. 8C show different views of an embodiment of a design of the key switch module 46 according to the present invention. Along these lines, Fig. 8A represents an exploded side view of the key switch module 46. Also, Fig. 8B represents an assembled side view of the key switch module 46.

Additionally, Fig. 8C contains an exploded top view of the key switch module 46. The functional components can include a key cap 120, key cap tabs 122, a switch 124, a switch plunger 126, alignment pins 128, switch contact leads 130, wires 132, right angle headers 134, header pins 136; a base 138; a shroud 140, and a foot 142. It should be remembered that this represents only one embodiment of a switch and associated elements that may be included in an embodiment of the present invention. Those of ordinary skill in the art would understand how to make modifications or substitutions of various components of the switch shown in Figs. 8A, 8B, and 8C without undue experimentation once aware of the disclosure contained herein.

According to one embodiment, the key switch modules 46 may be constructed as follows. A standard piece of epoxy-glass composite perforated circuit mounting board, such as board having 0.1-inch hole spacing, available from Digi-Key Corporation, may be cut to create the "four hole by four hole" square base 138. Two right angle headers 134, such as headers having 0.1 inch spacing and being gold plated and being available from Digi-Key Corporation, may be secured with epoxy adhesive to the base 138. The headers may be secured such that the header pins 136 extend through the central holes in the base 138, as shown in Fig. 8A and Fig. 8C.

The alignment pins 128 may be removed from the switch 124. One example of the switch is the ML series, normally-open, ultralow-profile key switch available from Cherry Corporation of Waukegan, IL. The switch contact leads 130 may be connected with the wires 132 to two diagonally opposed header pins at solder points 144A and 144B, as shown in Fig. 8C. The remaining two header pins 136 may not be connected to the switch contact leads 130. Rather, they may be included for extra structural support when the key switch module 46 is connected to the matrix 42.

The switch 124, base 138, and right angle headers 134 may be inserted into the center of the shroud 140. The foot 142 may then be applied to the bottom of the assembly. The components may be aligned such that the header pins 136 are centered in the circular opening of the foot 142 and the top of the key switch 124 rises higher than the lip of the shroud 140. Once the components are aligned, the remaining space within the shroud 140 may be filled. One example of a material that may be used to fill the space is epoxy, which would then need to cure. Finally, the key cap 120 may be added by inserting the key cap tabs 122 into the corresponding holes in the key switch plunger. Fig. 8B represents a side view of the assembled key switch module 46.

As stated above, the present invention permits switches to be arranged in any desired arrangement. Fig. 9 illustrates one possibility of this. Along these lines, Fig. 9 represents a top view showing three examples of how key switch modules 46 may be inserted into the pads 106. Footprint circles 146 are included in Fig. 9 to illustrate the approximate location of the header pins beneath the key switch modules 46.

Proper key switch module 46 placement typically requires that all four headers pins 136 are inserted fully into available tie points 96 of a single pad 106. The edge of the key switch module 46 may overlap the framing row 100, the spacer 104, the central divider 98, or the edge of the matrix 42. As shown in Fig. 9, it is even possible for the edge of the key switch module 46 to overlap another adjacent pad 106.

Furthermore, although only a single orientation is shown, the key switch module 46 also may be inserted into the pad 106 in any of four separate orthogonal rotations. Along these lines, a key switch module 46 may be rotated 90, 180, or 270 degrees around its vertical axis, as compared to the arrangements shown in Fig. 9. Geometrical constraints prevent more than one key switch module 46 from being placed into the same pad 106. Additionally, the framing row 100, the spacer 104, and the central divider 98 in the embodiment shown in Fig. 9 prevent a single key switch module 46 from connecting two separate pads 106.

The complimentary "diagonal" wiring of the header pins 136 and the "alternating row" wiring of the pads 106 means that however the key switch module 46 is positioned within a pad 106, depressing the key cap 120 will close a connection between two adjacent lines in the pad 106, connecting a row input line 108, shown in Fig. 6, to a column input line 110, shown in Fig. 6. This design permits the matrix scanner to use the pads 106 to identify uniquely any activated key switch module 46 properly inserted into the matrix 42.

The matrix 42 design described above and shown in Figs. 1, 5, 6, and 9 offers many advantages. Among the advantages are that the design includes significantly fewer individual nodes that a matrix scanner 78 must monitor. The flexibility of the key switch module 46 placement for the present invention may be substantially similar to a pure matrix design. However, by dividing the matrix 42 into separate pads 106, the matrix scanner 78 in this embodiment needs only to monitor 80 unique locations rather than individually monitoring all 2,800 of the open tie points 96. This allows use of readily available, and much slower, keyboard controller circuits, generally lowering the device cost and level of complexity.

Notwithstanding the above, Fig. 9 and the above description only illustrate one possible design of a matrix 42 and key switch modules 46. The description and illustration do not preclude other possible matrix designs. Along these lines, the matrix may include a matrix of individual tie points, each of which are mapped separately by a matrix scanner. Such a design could eliminate the need for pads 106 and framing rows 100. Those of ordinary skill in the art could determine alternative embodiments of the matrix, connections, and scanner without undue experimentation once aware of the disclosure contained herein.

As discussed above, the matrix scanner 78 can monitor the matrix 42 and transmit key actuation data to a microcontroller 80. One example of a microcontroller 80 that may be utilized with the present invention is a High-Speed Microcontroller Development system available from Systronix, Inc. of Salt Lake City, UT. However, any suitable microcontroller may be utilized.

The microcontroller includes a microprocessor. One example of a microprocessor that may be utilized according to the present invention is a Dallas Semiconductor Corporation 80C320 clocked at 33 MHz. Of course, any suitable microprocessor may be utilized.

The microcontroller 80 typically includes memory for storing various data for operation of the device. Along these lines, the microprocessor may include 128 kilobytes of non-volatile random access memory 88 and on-board input/output (I/O) pins and program loader logic. The memory 88 is retained during the power-off state with an on board capacitor. Of course, any type and amount of memory may be utilized. Possible arrangements and functions of the memory are described in greater detail below.

The block diagram in Fig. 10 illustrates the I/O pins available on the microcontroller 80. The names assigned to pins P1.0 through P3.7 correspond to the pin variable names used in the source code provided in Appendix A.

The functions of the microcontroller and associated microprocessor and memory may be carried out by the host computer.

Fig. 11 represents a schematic drawing that illustrates a display circuit in the form of an LCD circuit 90 and a mode selection and button control circuit 92. The LCD circuit 90 may include an interface. One example of an interface is a Hitachi 44780 interface, configured for a two-line by sixteen-character LCD 48 display, available from Systronix, Inc.

The mode selection and button control circuit 92 may include the left button 50, the right button 52, the label button 54, the clear button 68, and the mode switch 56. The four buttons typically are all normally open momentary push buttons. Such buttons are available, from Digi-Key Corporation, among other suppliers.

According to one embodiment, the mode switch 56 is a two-pole, six position rotary switch. Three positions may be locked out since they may not be required according to this embodiment. One example of such a switch is available from Radio Shack, a subsidiary of the Tandy Corporation of Fort Worth, Texas.

The microcontroller 80 can send data to the LCD circuit 90 and read data from the mode selection and button control circuit

92 through memory-mapped I/O controlled by a programmable logic device (PLD) address decoder chip 148. One example of such a chip is available from Systronix Inc., with part number ATF16V8CZ-15JC. Of course, as with any component described herein, this represents just one example of a chip that may be utilized according to the present invention.

The electrical schematic in Fig. 12 shows an embodiment of I/O circuits 82, 84, 86 and a system reset circuit 94 that may be included in an input device according to the present invention. As described previously, an input/output circuit A 82 may be utilized to transfer data to and from a matrix scanner 78. An input/output circuit B 84 may be included to transfer data to and from a standard keyboard 72. An input/output circuit C 86 may be employed to transfer data to and from a host computer 70.

Although not included in their respective dashed boxes, each of the three I/O circuits may also utilize a non-inverting transparent latch 150 for input to the microcontroller 80. One example of such a latch is a CMOS 74HCT573 Logic Device, available from Radio Shack Corporation. Each of the I/O circuits may use the non-inverting transparent latch 150 as an input buffer to read the status of the clock and data lines from the input devices.

The I/O circuits may each use a separate three-state line driver 152 for output from the microcontroller 80. One example of such a driver is a CMOS 74HCT244 Logic Device, available from Radio Shack Corporation. The three-state line driver 152 may be utilized to drive a corresponding digital clock and data lines low (logic zero) for output.

The system reset circuit 94 may be located inside the housing 40 and typically is not available to the user. The reset typically is used only after control software is loaded onto the microcontroller 80 to initiate a memory initialization routine. The "System Reset" routine of the source code listing in Appendix A provides more information on the memory initialization routine.

As stated above, the hardware described above may be directed by a microcontroller 80. The microcontroller may execute commands of the control software stored in the memory 88, also referred to above. The memory 88 may be split into a block of 64 kilobytes of code memory 154, as shown in Fig. 13., and a block of 64 kilobytes of data memory 156, as shown in Fig. 14. Referring to Fig. 13, the code memory 154 may store a main program code 158, lookup tables 160, an interrupt one (INT1) vector 162, an interrupt zero (INT0) vector 164, and a reset vector 166.

The main program code 158 can include all the instructions needed by the microcontroller 80 to operate a device according to the present invention. The instructions can include the interrupt service routines that retrieve data from the standard keyboard 72 (via INT1) and the matrix scanner 78 (via INT0). Lookup tables 160 may be used by the control software to retrieve from the data memory 156 data associated with a programmed pad 106. The two interrupt vectors may redirect the program instruction pointer to the appropriate interrupt service routines whenever an interrupt is generated by activity on the matrix scanner 78 or the standard keyboard 72 clock lines. The reset vector 166 may direct the program instruction pointer to the start of the control software at power up or restart of the microcontroller 80.

A complete listing of an embodiment of control software source code that may be utilized according to the present invention is provided in Appendix A. The source code is a hybrid of BASIC and Assembly Language. BASIC is used for the overall program flow control and string manipulation while in-line Assembly language is used for speed sensitive operations. The source code was compiled using the BCI51™ PRO BASIC Cross Compiler, Version 1.40, available from Systronix, Inc. Details for transferring the compiled control software to the code memory 154 are provided with documentation associated with the Cross Compiler. Of course, if the functions carried out by the controller are accomplished by a host computer, then the discussion herein of the controller, memory, software and other associated elements does not apply.

Fig. 14 represents a block diagram showing an allocation of the data memory 156. Memory-mapped I/O addresses 168 may reside at the top four kilobytes of the data memory 156 and may be used to access the LCD circuit 90 and the mode selection and button control circuit 92. A macro buffer 170 may temporarily store macros as they are generated by a user until the macros are assigned to a particular pad 106 location.

A data set label bank 172, may hold the strings identifying the four matrix data sets 174 available for use. The matrix data sets 174 can effectively provide four completely independent key configurations. For example, a user may use one matrix data set 174 for a spreadsheet application and change to another matrix data set 174 for a word processing application.

Each of the four matrix data sets 174 can provide 126 bytes of storage for each pad 106 in the matrix 42. The user may use one of the matrix data sets 174 at a time and may change between them using the left button 50 and right button 52, as shown in Fig. 1 and Fig. 11. Keyboard buffer 176, matrix buffer 178, and send buffer 180 may each provide a 256 byte circular buffer to store and transfer the standard keyboard 72 data and the matrix scanner 78 data to the host computer 70. The bottom sixteen kilobytes may be reserved for variables defined in the control software.

Flow diagrams illustrated in Figs. 15-19 present an example of a general overview of an embodiment of microcontroller 80 control software operation. It should be noted that these flow diagrams only portray the functions necessary to convey a broad understanding of one embodiment of the present invention. The full source code documentation, provided in Appendix A, reveals more detail about the control software operation according to one embodiment for carrying out these functions. Software functions not discussed in the flow diagrams shown in Figs. 15-19, but included in the source code, include data collision detection; keyboard status light operation, such as caps lock, number lock, and scroll lock; data storage address calculation; and memory coding and storage schemes.

The flow diagram of Fig. 15 shows the start-up process of the microcontroller 80. At power on, the microcontroller 80 may dimension the system variables 182, initialize the system settings 184, including the display 48 and I/O ports, and enable the interrupts 186. Next, the microcontroller may read the program mode 188 indicated by the mode switch 56. The microcontroller 80 may branch to the software routine for one of the three possible modes, a run mode 192, shown in Fig. 17, a key mode 194, shown in Fig. 18, or a macro mode 196, shown in Fig. 19.

Two interrupts (INT0 and INT1) may be utilized according to the present invention. Along these lines, INT0 may be connected to the clock line of the matrix scanner 78. Additionally, INT1 may be connected to the clock line of the standard keyboard 72. When data traffic is sent from the matrix scanner 78 or the standard keyboard 72, triggered by a falling edge on the clock line, the main program execution may be paused while the program jumps to the calling interrupt service routine to process the interrupt.

The flow diagram shown in Fig. 16 illustrates an embodiment of a process that may be followed during an interrupt service routine, or ISR 190. Each time the ISR 190 is called, another bit from the data traffic may be acquired and stored in a temporary variable. Once an entire byte of data is received, the value may be transferred from the temporary variable to the end of the appropriate circular buffer, and the temporary variable may be cleared. Data captured by INT0 may be transferred to the matrix buffer 178. On the other hand, data captured by INT1 may be transferred to the keyboard buffer 176.

The flow diagram shown in Fig. 17 illustrates an embodiment of operation of the microcontroller 80 during the run mode 192. The run mode 192 is the functional mode for the device. As shown in the diagram, the microcontroller 80 may repeatedly loop through the run mode routine checking the contents of the matrix buffer 178, the keyboard buffer 176, and the send buffer 180. During this looping process, if the user activates a key switch module 46 on the matrix 42, the matrix scanner 78 may transmit the corresponding byte to the microcontroller 80, triggering INT0 and pausing the run mode loop execution. Once the ISR 190 has received the byte and added it to the end of the matrix buffer 178, the microcontroller 80 may resume normal program execution. In this manner, data may be added to the end of the circular matrix buffer 178.

When the microcontroller 80 subsequently detects a new byte in the matrix buffer 178, it may transfer a previously stored input function, such as a single key data or macro data, to the send buffer 180. This input function may be fetched from the memory 88 and may be determined by both an active matrix data set 174 and the pad 106 identified by this newly added byte. After handling the matrix buffer 178, the microcontroller 80 may perform a similar operation for the keyboard buffer 172. However, unlike the conversion process applied to the data in the matrix buffer 178, data in the keyboard buffer 172 may be relayed directly to the send buffer 180.

The microcontroller 80 may next examine the contents of the send buffer 180. Then, the microcontroller may transfer newly added data from the matrix buffer 178 and/or the keyboard buffer 172 to the host computer 70. Finally, the microcontroller 80 may check for a button press or a mode change before starting the loop again.

Using the interrupt service routines 190 and the three circular buffers as described allows for the simultaneous use of the standard keyboard 72 with the present invention. For some known keyboard wedge devices, pressing and holding down a key on the standard keyboard 72 blocks data from being sent by the secondary device until the standard keyboard stopped transmitting data, that is, until the key is released. By applying interrupts and circular buffers, data traffic from both the standard keyboard 72 and the matrix scanner 78 may be detected, captured, multiplexed, and forwarded to the host computer 70 effectively.

The flow diagram shown in Fig. 18 illustrates the operation of the microcontroller 80 during the key mode 194. The key mode 194 is a programming mode for the device, allowing the user to program the key switch modules 46 inserted into the matrix 42 to operate like standard keyboard keys. For example, during the key mode 194, a user may program a key switch module 46 as the "Q" key of the standard keyboard 72.

After returning to the run mode 192, the programmed key switch module 46 mimics the electronic behavior of a standard "Q" key. Depressing the key switch module 46 generates a "Q-key make-code" to be sent to the host computer 70. After a brief delay, holding the key switch module 46 down causes the "make code" to be sent repeatedly, duplicating the key repeat feature of the standard keyboard 72. When the key switch module 46 is released, a "Q-key break code" is sent to the host computer 70. The host computer 70 does not detect that the transmitted key input function is not generated by a standard keyboard 72.

The key mode 194 operation shares many similarities to the run mode 192 operation. However, a major difference is that no data typically is transmitted to the send buffer 180, or the host computer 70, during the key mode 194. As shown in the flowchart illustrated in Fig. 18, during the key mode loop, the microcontroller 80 may monitor the data traffic sent by the standard keyboard 72 and retains the value of the last key transmitted. Once a key switch module 46 is pressed, indicated by a new data appearing in the matrix buffer 178, the value of the last key transmitted by the standard keyboard 72 may be stored in the current matrix data set 174 memory location for the pad 106 in which the key switch module 46 is placed. The value of the last key transmitted and the assigned pad location may be displayed by display 48.

The flow diagram shown in Fig. 19 illustrates an embodiment of the operation of the microcontroller 80 during the macro mode 196. Like the key mode 194, the macro mode 196 is a programming mode for the device. However, the macro mode 196 allows the user to assign key macros to a key switch module 46. A macro can include a sequence of multiple key presses and/or other inputs from other computer input devices, such as a mouse, joystick, trackball, game controller or other input device. A difference between a key assignment and a macro assignment is that after returning to the run mode 192, a key switch module 46 may be programmed with a macro that mimics multiple sequential key presses of a standard keyboard 72 or other input of any other input device.

According to one example, instead of a single key, a macro can include a string of keys, such as "dog". A macro can also initiate common control key sequences, such as the "Ctrl" + "S" key combination. Any other string of inputs may also be included in a macro. Such macros may find use in many common computer applications.

In each of the four matrix data sets 174, each pad 106 in the matrix 42 may have memory allocated for macro sequences up to 126 bytes long. During the run mode 192, a key switch module 46 programmed with a macro transmits the entire stored key sequence when pressed. Typically, nothing is transmitted when the key switch module 46 is released.

As shown in the flow chart illustrated in Fig. 19, during the macro mode 196, the microcontroller 80 may monitor and store data traffic from a standard keyboard 72 through the keyboard buffer 176. The microcontroller 80 may continue to add the keyboard traffic to the recorded macro until either a key switch module 46 is pressed or the macro becomes too large. Once a key switch module 46 is pressed, which may be indicated by a new data appearing in the matrix buffer, the value of the recorded macro may be stored in the current matrix data set 174 memory location for the pad 106 in which the key switch module 46 is placed. To make it easier for the user to monitor the macro as it is recorded, data traffic from the standard keyboard 72 may be transmitted to the host computer 70 during the macro mode 196.

To operate a device according to the present invention, the device may be connected to a host computer 70 and a standard keyboard 72, as shown in Fig. 3. This description particularly applies to the embodiment shown in Figs. 1 and 2. The user may then arrange one or more key switch modules 46 into a desired layout on the matrix 42, as shown in Fig. 1.

Assuming that the user has programmed the desired input functions for the pads 106 into which the key switch modules 46 are inserted, the device may be set to the run mode 192 with the mode switch 56 pointing to "Run" as shown in Fig. 20A. During the run mode 192, a first display line 198 of display 48 may include a matrix data set number, or set number 202 and a matrix data set label, or set label 204. The set number 202 may communicate the currently selected matrix data set 174, numbered one to four. The set label 204 may be a user defined string that identifies the name of the current matrix data set 174. To change the current set label 204, a user may press the label button 54, type a new set label 204 using the standard keyboard 72 letter and number keys, and then accept the change by pressing the "Enter" key on the standard keyboard 72.

During the run mode 192, the user also may change to the next or previous matrix data sets 174 using the left button 50 and the right button 52. The ability to change the data set 174 allows the user to assign, store, and use separate input functions for separate applications and key switch module 46 layouts.

Additionally during the run mode, standard keyboard 72 data traffic may be relayed through the device to the host computer 70. The user may use the standard keyboard 72 in a normal manner. The user may also use the programmed key switch modules 46 for computer input functions.

Pressing a key switch module 46 will send the programmed function, whether a single key or action or macro of a plurality of actions, of the corresponding pad 106 location to the host computer 70. Pressing a key switch module 46 that has not been programmed for the current matrix data set 174 will not have any effect. In other words, no data will be transmitted to the host computer 70. The present invention permits a user to add the benefit of a programmable, dynamic computer input device while keeping the familiar standard keyboard 72.

For the currently selected matrix data set 174, the user may overwrite an existing input function assignment or create a new assignment if none already exists using either the key mode 194 or the macro mode 196. As described earlier, the key mode 194 can assign the function of a single key to the pad 106 of a key switch module 46. The macro mode 196 may assign key macros to the pad 106 of a key switch module 46.

The device may be set to the key mode 194 by turning the mode switch 56 to "Key" as shown in Fig. 20B. During the key mode 194, the display 48 may change the display to indicate the current programming status of the device. The first display line 198 may show the last key, if any, the user pressed on the standard keyboard 72. The second display line 200 may show the position of the pad 106 that was assigned with the last key detected from the standard keyboard 72.

The pad 106 positions may be identified with an alphanumeric grid. In other words, columns in the matrix 42 may be labeled A through J, while rows may be labeled one through eight. To assign a key function, the user typically first ensures that a key switch module 46 is placed within the desired pad 106. Next, the key to be assigned is pressed on the standard keyboard 72 or other input device. Then, the key switch module 46 may be pressed to bind the detected key function to the corresponding pad 106.

According to one example, a user desires to assign the pressing the "Q" key on a keyboard as the function to a key switch module 46 inserted in the pad 106 located at position A5. First, the user may set the device to the key mode 194 with the mode switch 56. Next, the user may insert a key switch module 46 into the desired pad 106. Then, the user may press the "Q" key on the standard keyboard 72. This key press may be detected by the device and may be indicated on the first display line 198. Next, the user may press the key switch module 46, causing the key function to be bound to the corresponding pad 106. The location of the assigned pad 106 may be indicated on the second display line 200. This process can be repeated until all the desired key switch modules 46 are programmed.

The device may be set to the macro mode 196 by turning the mode switch 56 to "Macro" as shown in Fig. 20C. The macro mode 196 is similar to the key mode 194, except that instead of storing just the last key pressed on the standard keyboard 72 and/or other input from any other computer input device,. a sequence of one or more key presses or other inputs may be recorded and assigned to a single pad 106 on the matrix 42.

During the macro mode 196, display 48 may change the display to indicate the current programming status of the device. The first display line 198 and the second display line 200 can show the current number of data bytes recorded for the current macro. To record a macro with the current embodiment, the user may type keys in a desired sequence on the standard keyboard 72 . The input device of the present invention can record these key sequences in the macro buffer 170, one embodiment of which is shown in Fig. 14, until the macro reaches maximum length. After reaching the maximum length, the device ceases to record any further additions. The maximum length may vary, depending upon the embodiment.

Once the macro has been recorded, the user may then press the desired key switch module 46 to assign the macro to a corresponding pad 106. The second display line 200 may then change to indicate the position of the pad 106 assigned with the macro. Once the macro has been assigned, the user may continue to record other macros or may return the device to the run mode 192.

During the run mode 192, a selected matrix data set 174 may be erased using the clear button 68. After pressing the clear button 68, the display 48 may display a confirmation message to prevent accidental erasure, as shown in Fig 20D. Selecting "No", the right button 52 in the illustrated embodiment may skip the erase command and return to the run mode 192. Selecting "Yes", the left button 50 in the illustrated embodiment, can confirm the erase command and clears any input functions, whether single function or macro of functions, assigned to the pads 106 for the selected matrix data set 174. The matrix data set label 204 may be changed to read "Empty". The device may then return to the run mode 192. Erasing a matrix data set 174 can permit the user to reprogram the device easily and effectively as the need arises.

Many alternate methods also exist for identifying or programming the key switch modules 46 that may also be applied to the present invention. For example, the key switch modules 46 may be attached to external connection wires 208 as shown in Fig. 22. To program the key switch module, the external connection wire 208 may be removably attached to an input function post 210 that represents a specific key or macro. According to this embodiment, the matrix 42 itself does not need to be scanned. Rather, the matrix may be used as a mechanical platform to hold the key switch modules 46. Only the input function posts need to be monitored to detect activity from the key switch module 46.

Other potential matrix-switch design combinations include producing key switch modules 46 that contain individual network identifiers that are transmitted to a common matrix bus. This can allow the key switch module 46 to identify itself instead of relying on its position within the matrix 42. This design can eliminate the traditional matrix scanner by substituting an electrical bus mastering and monitoring circuit.

According to another embodiment, lookup tables 160 and the matrix data sets 174 are moved to the host computer 70. In this embodiment, the host computer 70 assumes the task of assigning the programmable functions of individual matrix positions after receiving static functions from the device in response to switch activation. Using a host computer to re-assign functions to specific keys of the standard keyboard is well known in the art. Although this embodiment may require a modification to the host computer 70, such as changing the key map configuration, it may result in a less complicated design of the hardware and software of the present invention.

The host computer 70 may also provide a means to assign, store, and transfer key map configurations for the present invention. While the present invention may include a means for programming the device using the standard keyboard 72, this programming method may be augmented or replaced with software running on the host computer that allows users to select a function for a particular key position. That information could then be transferred to a device according to the present invention. The transfer could be accomplished by any means, such as by using a serial, parallel, USB, IR, or other connection.

The present invention may include a bypass circuit to permit a standard keyboard 72 or other input device to remain connected and operational when the present invention is not powered. Fig. 25 illustrates one possible embodiment of a bypass circuit 214 that may be utilized with the present invention. Of course, if the present invention is not connected between a keyboard and a host computer, then such a circuit may not be necessary.

Without a bypass circuit, keyboard traffic, for example, may not be relayed to a host computer 70 when the device according to the present invention is turned off. The embodiment of the bypass circuit 214 shown in Fig. 25 includes four reed relays 216 to connect the standard keyboard 72 directly to the host computer 70 automatically when power is removed from the device. When the power is switched on, the reed relays 216 reconnect the standard keyboard 72 and the host computer 70 to the appropriate I/O circuits as described in the basic embodiment.

Fig. 27 illustrates an embodiment of a system according to the present invention that includes wireless communication between elements of the system. Along these lines, Fig. 27 illustrates a plurality of switches 226 that utilize wireless communication techniques, such as radio frequency and/or infrared transmission to indicate their activation status to a receiving unit 228. Any wireless communication protocol may be utilized.

An attachable backing included on the wireless switches 226 can permit a user to removably or permanently attach individual switches to a variety of useful locations. Any attachable backing could be utilized. Along these lines, one or more adhesives, suction, or magnets may be utilized. Surfaces that the at least one switch could be attached to include a monitor 230; case of a computer, including host computer 70; a desk or other work surface 232; a computer keyboard, including the standard keyboard 72; a switch tray or platform 234; a chair (not shown); a computer mouse 236; and a printer (not shown). Any other surface may also be utilized.

In view of the disclosure contained herein, the present comfortable computer input device for many common applications. In addition to the dynamic positioning of individual keys, the invention can rely on standard interface protocols, thereby eliminating the needs for special software drivers to be installed on the host computer. Furthermore, the device can operate in conjunction with standard input devices, such that a user may continue to use these standard devices in a normal manner.

The foregoing description of the invention illustrates and describes the present invention. Additionally, the disclosure shows and describes only the preferred embodiments of the invention, but as aforementioned, it is to be understood that the invention is capable of use in various other combinations, modifications, and environments and is capable of changes or modifications within the scope of the inventive concept as expressed herein, commensurate with the above teachings, and/or the skill or knowledge of the relevant art. The embodiments described hereinabove are further intended to explain best modes known of practicing the invention and to enable others skilled in the art to utilize the invention in such, or other, embodiments and with the various modifications required by the particular applications or uses of the invention. Accordingly, the description is not intended to limit the invention to the form disclosed herein. Also, it is intended that the appended claims be construed to include alternative embodiments.

Claims

I claim:

1. A configurable computer input device, comprising:

at least one switch removably attachable to a surface and in communication with a processor, at least one function being assignable to activation of the switch, wherein the at least one switch is repositionable at distances smaller than a length or width of the at least one switch;

circuitry in communication with the at least one switch for assigning at least one function to activation of the switch;

circuitry for communicating the at least one function to a host computer; and

circuitry for determining the actuation status of the at least one switch and communicating the actuation status to the processor.

2. The device according to claim 1, wherein the surface that the at least one switch is removably attachable to includes a surface on a structure selected from the group consisting of a mouse, a monitor, a keyboard, a desk, a work surface, a keyboard tray, a switch tray, a switch platform, a chair, a computer, and a printer.

3. The device according to claim 1, wherein the at least one switch and the circuitry for determining the actuation status

of the at least one switch are in wireless communication.

4. The device according to claim 1, wherein the device is in wireless communication with the host computer.

5. The device according to claim 1, wherein the surface that the at least one switch is attached to comprises at least one matrix of receptacles that the at least one switch is removably attachable to at a plurality of positions, the device further comprising:

a memory for storing data comprising a mapping of the at least one function assigned to the at least one switch to the position of the at least one switch in the at least one matrix of receptacles.

6. The device according to claim 5, wherein the at least one switch comprises at least one pin that extends from the switch and is receivable by the matrix of receptacles.

7. The device according to claim 1, wherein the at least one switch comprises at least one of a keyboard key, a button switch, a keyboard key comprising a swiveling key-cap top, a keyboard key comprising a tilting key-cap top, a keyboard key comprising a swiveling and tilting key-cap top, and a keyboard key having a non-standard shape and size.

8. The device according to claim 5, wherein the at least

one matrix of receptacles is housed in a keyboard housing with a standard keyboard.

9. The device according to claim 5, wherein the at least one matrix of receptacles is housed in a videogame controller.

10. The device according to claim 1, wherein the at least one function comprises at least one electronic, alphanumeric, or keyboard function.

11. The device according to claim 10, wherein the at least one function comprises at least one function selected from the group consisting of movement of a joystick, actuation of a joystick button, movement of a mouse, actuation of a mouse button, actuation of a game controller, and actuation of a keyboard key.

12. The device according to claim 11, wherein a plurality of functions are non-permanently assignable to the at least one switch, including at least one of timing, cadence, and sequence of the functions.

13. The device according to claim 1, wherein the circuitry for determining the actuation status of the at least one switch scans the at least one switch.

14. The device according to claim 1, wherein the processor

comprises a controller for receiving the actuation status of the at least one switch, determining the function assigned to activation of the switch, and transmitting the at least one function to the circuitry for communicating the at least one function to the host computer.

15. The device according to claim 1, further comprising:
a memory for storing data comprising the at least one function assigned to the at least one switch.

16. The device according to claim 1, comprising at least two switches, wherein the plurality of keys may be positioned such that the at least two keys are not aligned.

17. The device according to claim 1, wherein the at least one switch is operable simultaneously with another computer input device in communication with the host computer.

18. The device according to claim 5, wherein the matrix of receptacles is non-contiguous.

19. The device according to claim 5, wherein the matrix of receptacles is non-planar.

20. The device according to claim 1, wherein the at least one switch comprises a switching mechanism, an attached key cap, and a switch housing.

21. The device according to claim 1, wherein the function is non-permanently assignable to the switch.

22. A configurable computer input device, comprising:

at least one switch removably attachable to a receiving surface;

at least one receiving surface for receiving the at least one switch at a plurality of positions, wherein a distance separating any two positions may be smaller than a length or a width of the at least one switch;

a memory for storing data comprising a mapping of at least one electronic, alphanumeric, or keyboard function to the at least one switch when the at least one switch is removably attached to a position of the receiving surface;

circuitry for scanning the at least one switch when the at least one switch is removably attached to a position of the receiving surface and for determining the actuation status of the at least one switch;

a controller responsive to the circuitry for scanning and for consulting the memory to obtain the at least one function mapped to the at least one switch upon actuation of the at least one switch; and

circuitry for transferring the at least one function obtained by the controller to a host computer with which the device communicates.

23. The device according to claim 22, wherein a keycap is

attached to the at least one switch.

24. The device according to claim 22, wherein a button is attached to the at least one switch.

25. The device according to claim 22, wherein the at least one receiving surface comprises a matrix of receptacles for receiving pins attached and electrically connected to the at least one switch.

26. The device according to claim 22, wherein the at least one switch comprises a switching mechanism, an attached key cap, and a switch housing.

27. A computer system, comprising:

a configurable computer input device, comprising at least one switch removably attachable to a surface and in communication with a processor, the surface comprising at least one matrix of receptacles that the at least one switch is removably attachable to at a plurality of positions, the at least one switch may be repositioned on the matrix of receptacles at distances smaller than a length or width of the at least one switch, at least one function comprising at least one electronic, alphanumeric or keyboard function is assignable to activation of the switch; circuitry in communication with the at least one switch for assigning at least one function to activation of the switch; circuitry for communicating the at least one function to a host computer; a memory for storing data comprising a mapping of the at least one function assigned to the at least one switch to the position of the at least one switch in the at least one matrix of receptacles; and circuitry for determining the actuation status of the at least one switch and communicating the actuation status to the processor; and

a host computer selected from the group consisting of a microcomputer and a video game computer.

28. A method for generating input to a computer, the method comprising:

providing a configurable computer input device comprising at least one switch removably attachable to a surface and in communication with a processor, at least one function being assignable to activation of the switch, wherein the at least one switch may be repositioned at distances smaller than a length or width of the at least one switch; circuitry in communication with the at least one switch for assigning at least one function to activation of the switch; circuitry for communicating the at least one function to a host computer; and circuitry for determining the actuation status of the at least one switch and communicating the actuation status to the processor;

providing at least one input from an existing computer input device;

recording the at least one input from the existing computer input device; and

assigning the at least one input from the existing computer input device to the at least one switch.

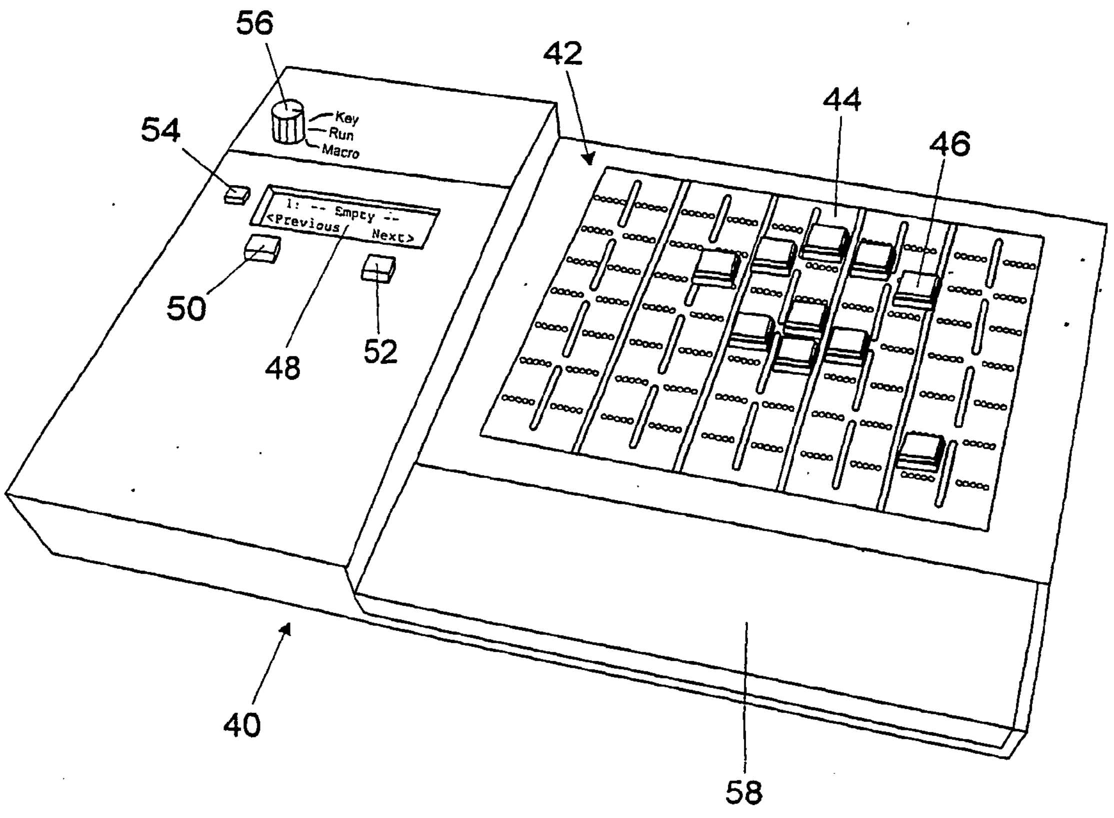


Fig. 1

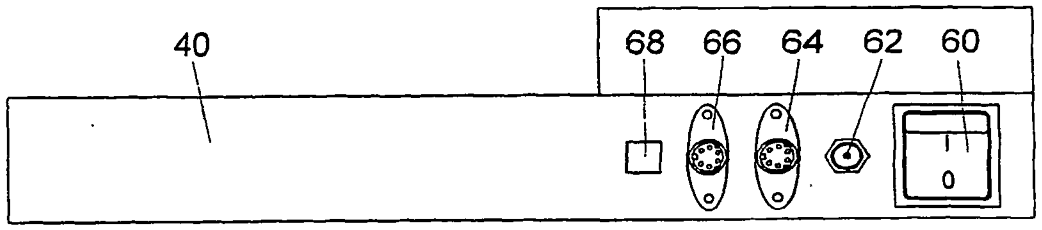


Fig. 2

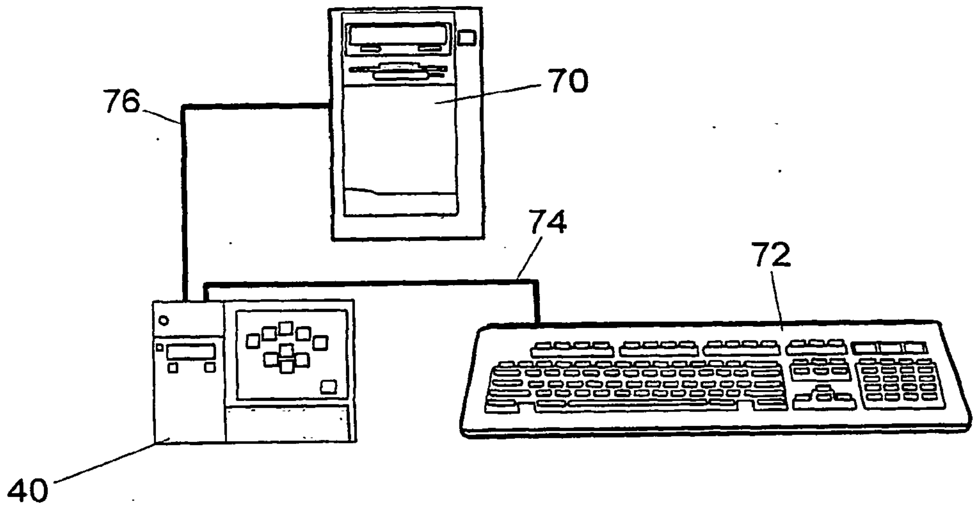
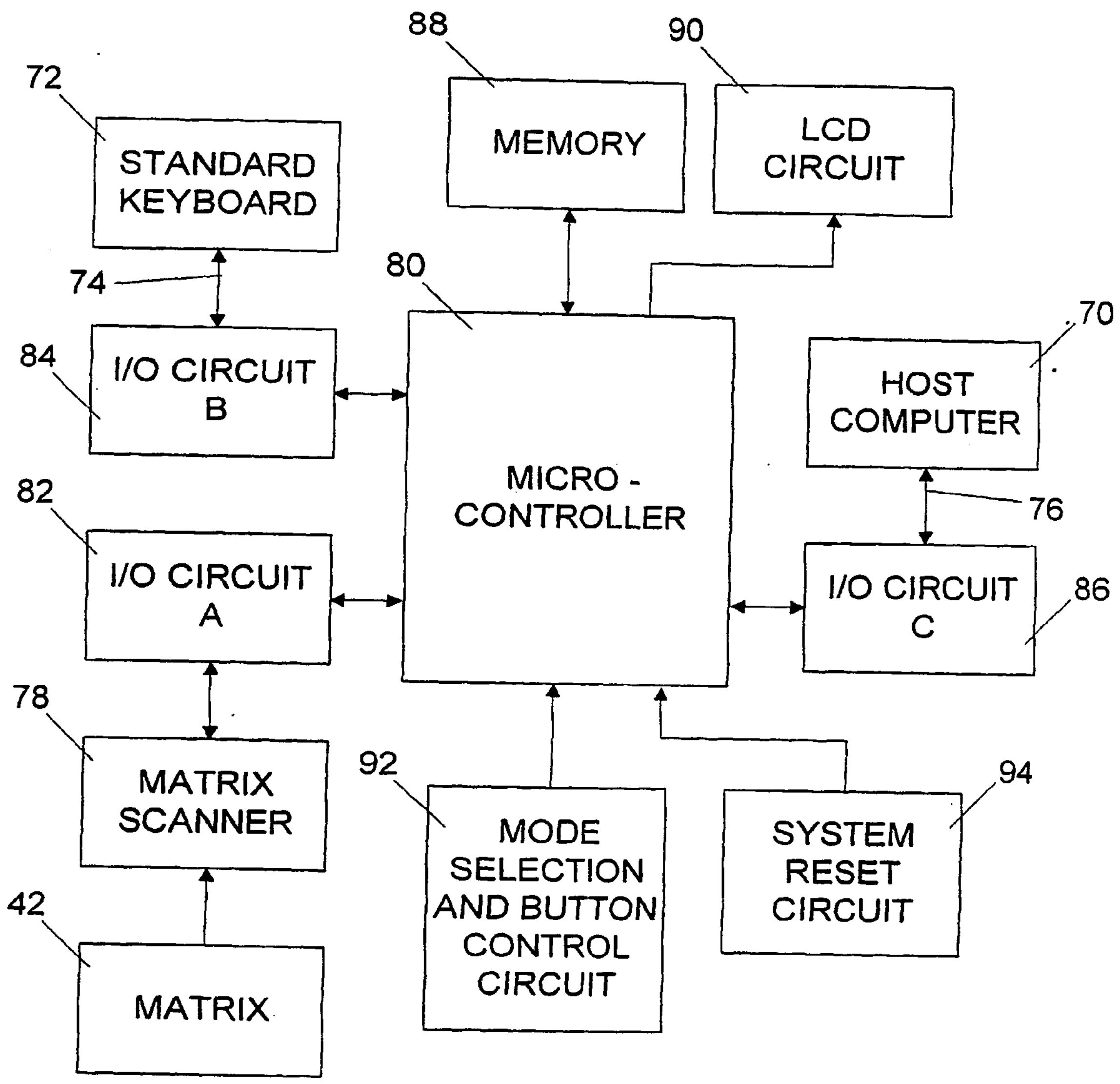


Fig. 3



88
90
72
MEMORY
LCD CIRCUIT
STANDARD KEYBOARD
74
80
84
I/O CIRCUIT B
70
HOST COMPUTER
MICRO - CONTROLLER
82
76
I/O CIRCUIT A
I/O CIRCUIT C
86
78
MATRIX SCANNER
92
94
MODE SELECTION AND BUTTON CONTROL CIRCUIT
SYSTEM RESET CIRCUIT
42
MATRIX

Fig. 4

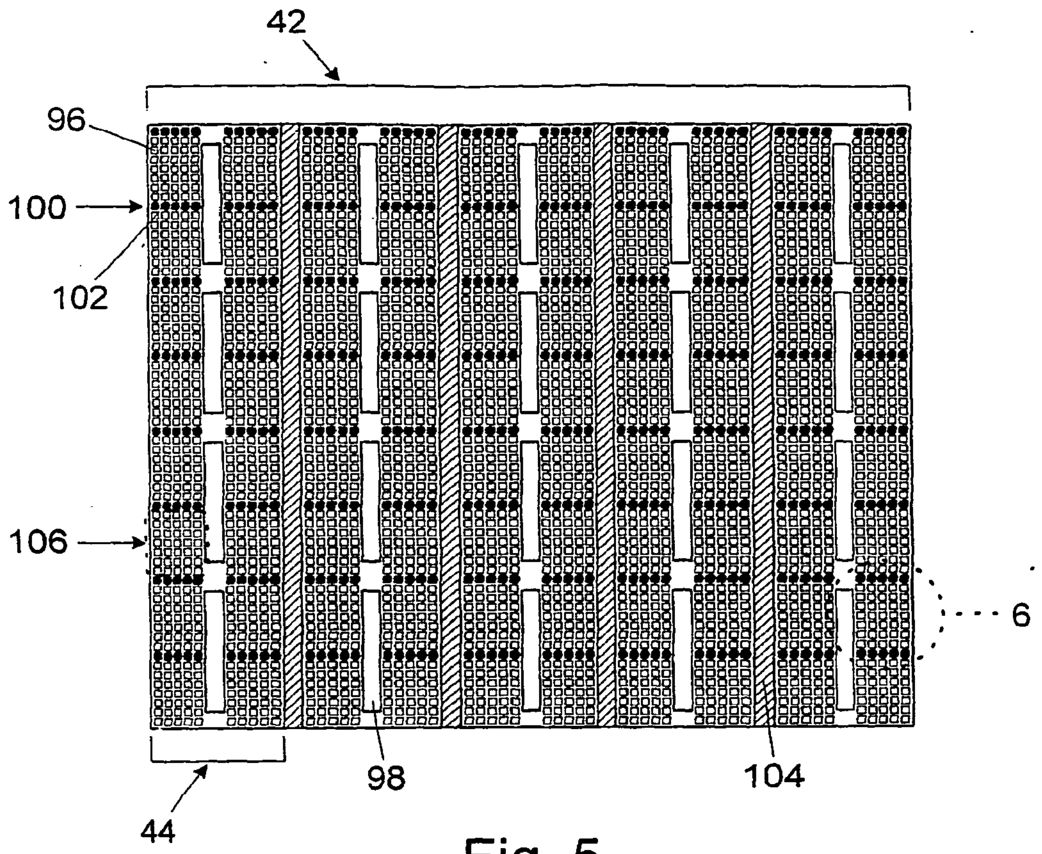

42
96
100
102
106
6
98
104
44

Fig. 5

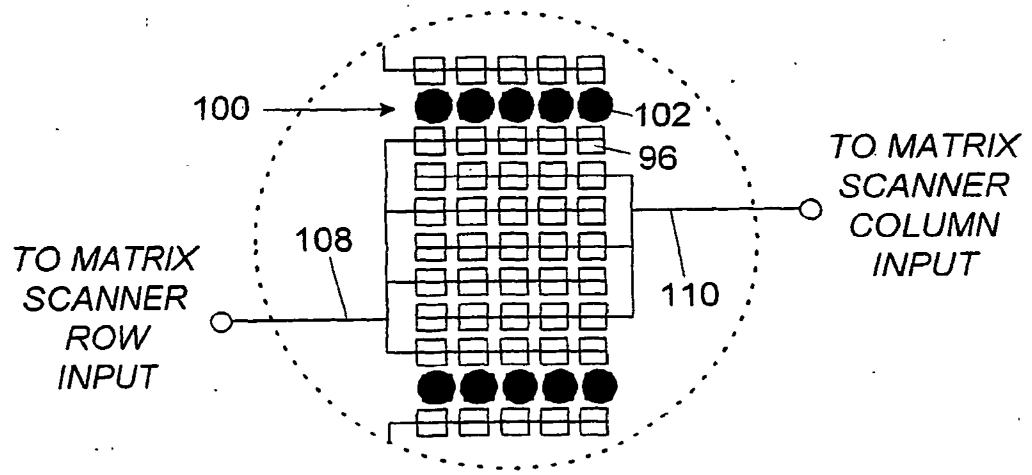

100
102
96
108
110
TO MATRIX SCANNER ROW INPUT
TO MATRIX SCANNER COLUMN INPUT

Fig. 6

Fig. 7

112 114 116 118

9 46 R : 2 C : 4	- 4E R : 1 C : 3	; 4C R : 0 C : 3	CAPS 58 R : 4 C : 10	F3 04 R : 4 C : 8	I 43 R : 3 C : 5	U 3C R : 3 C : 6	M 3A R : 6 C : 6	PAD * 7C R : 6 C : 15	R 2D R : 3 C : 7
ESC 76 R : 5 C : 11	F 2B R : 0 C : 7	F1 05 R : 1 C : 10	F10 09 R : 2 C : 2	PAD 2 72 R : 0 C : 12	PAD 3 7A R : 0 C : 15	PAD 4 6B R : 4 C : 9	PAD 5 73 R : 4 C : 12	X 22 R : 6 C : 10	Y 35 R : 4 C : 6
Z 1A R : 6 C : 11	D 23 R : 0 C : 8	E 24 R : 3 C : 8	RET 5A R : 6 C : 2	PAD . 71 R : 5 C : 15	PAD + 79 R : 3 C : 16	PAD 0 70 R : 5 C : 12	PAD 1 69 R : 0 C : 9	V 2A R : 6 C : 7	W 1D R : 3 C : 10
= 55 R : 1 C : 5	A 1C R : 0 C : 11	B 32 R : 7 C : 7	BACK 66 R : 4 C : 2	N 31 R : 7 C : 6	O 44 R : 3 C : 4	P 4D R : 3 C : 3	PAD - 7B R : 7 C : 15	T 2C R : 4 C : 7	Tab 0D R : 4 C : 11
/ 4A R : 7 C : 3	[54 R : 4 C : 3	] 5B R : 4 C : 5	~ 0E R : 1 C : 11	K 42 R : 0 C : 5	L 4B R : 0 C : 4	L ALT 11 R : 5 C : 1	L CTR 14 R : 1 C : 17	SCR 7E R : 3 C : 1	SPA 29 R : 7 C : 2
8 3E R : 2 C : 5	' 52 R : 5 C : 3	\\ 5D R : 0 C : 2	. 49 R : 6 C : 4	NUM 77 R : 6 C : 9	G 34 R : 5 C : 7	H 33 R : 5 C : 6	J 3B R : 0 C : 6	Q 15 R : 3 C : 11	S 1B R : 0 C : 10
4 25 R : 2 C : 7	5 2E R : 1 C : 7	6 36 R : 1 C : 6	7 3D R : 2 C : 6	F5 03 R : 5 C : 2	F6 0B R : 5 C : 5	F7 83 R : 4 C : 4	F8 0A R : 1 C : 4	PAD 8 75 R : 3 C : 12	PAD 9 7D R : 3 C : 15
0 45 R : 2 C : 3	1 16 R : 2 C : 11	2 1E R : 2 C : 10	3 26 R : 2 C : 8	F11 78 R : 2 C : 9	F12 07 R : 2 C : 12	F2 06 R : 1 C : 8	F4 0C R : 5 C : 8	PAD 6 74 R : 4 C : 15	PAD 7 6C R : 3 C : 9

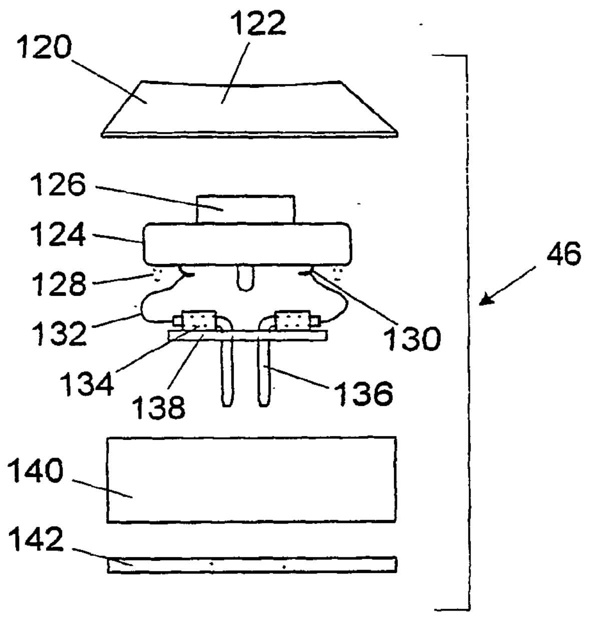

120
122
126
124
128
132
134
138
130
136
140
142
46

Fig. 8A

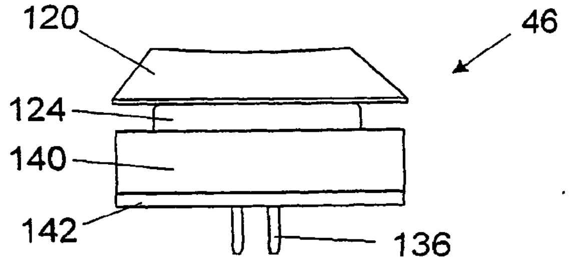

120
46
124
140
142
136

Fig. 8B

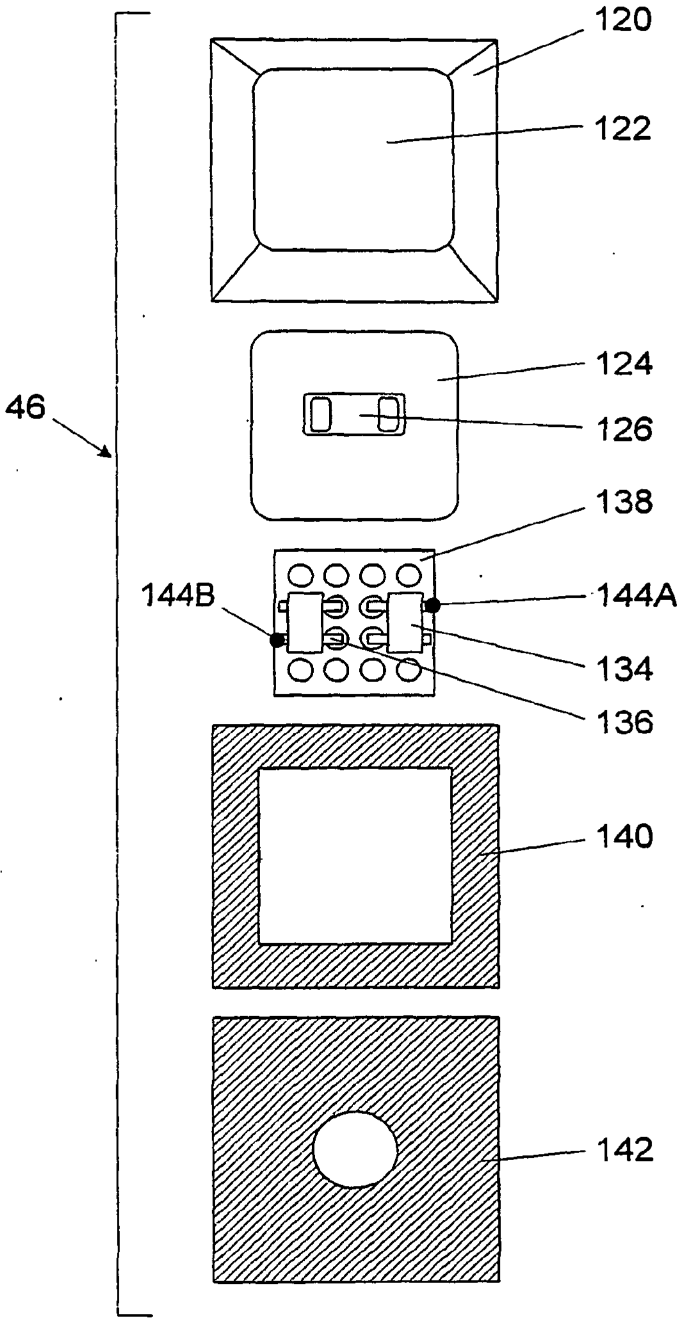


Fig. 8C

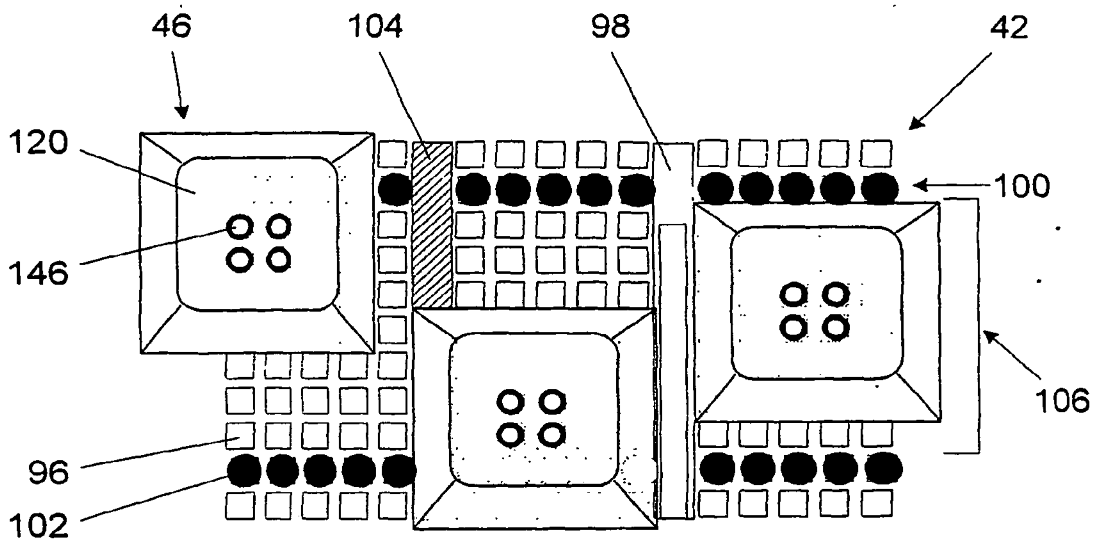


Fig. 9

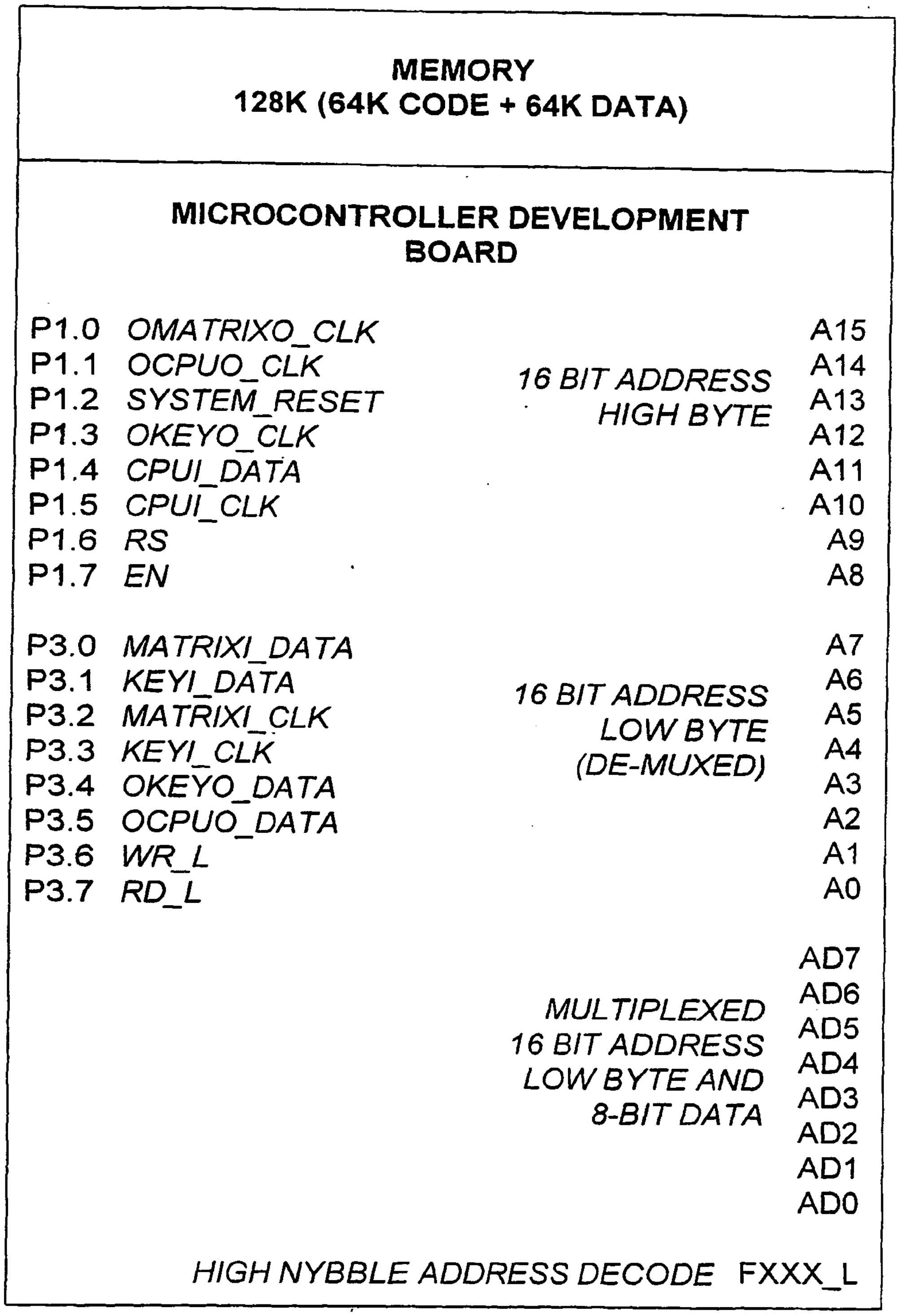

MEMORY
128K (64K CODE + 64K DATA)
MICROCONTROLLER DEVELOPMENT BOARD
P1.0 OMATRIXO_CLK
P1.1 OCPUO_CLK
P1.2 SYSTEM_RESET
P1.3 OKEYO_CLK
P1.4 CPUI_DATA
P1.5 CPUI_CLK
P1.6 RS
P1.7 EN
P3.0 MATRIXI_DATA
P3.1 KEYI_DATA
P3.2 MATRIXI_CLK
P3.3 KEYI_CLK
P3.4 OKEYO_DATA
P3.5 OCPUO_DATA
P3.6 WR_L
P3.7 RD_L
16 BIT ADDRESS HIGH BYTE
A15
A14
A13
A12
A11
A10
A9
A8
16 BIT ADDRESS LOW BYTE (DE-MUXED)
A7
A6
A5
A4
A3
A2
A1
A0
MULTIPLEXED 16 BIT ADDRESS LOW BYTE AND 8-BIT DATA
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
HIGH NYBBLE ADDRESS DECODE FXXX_L

Fig. 10

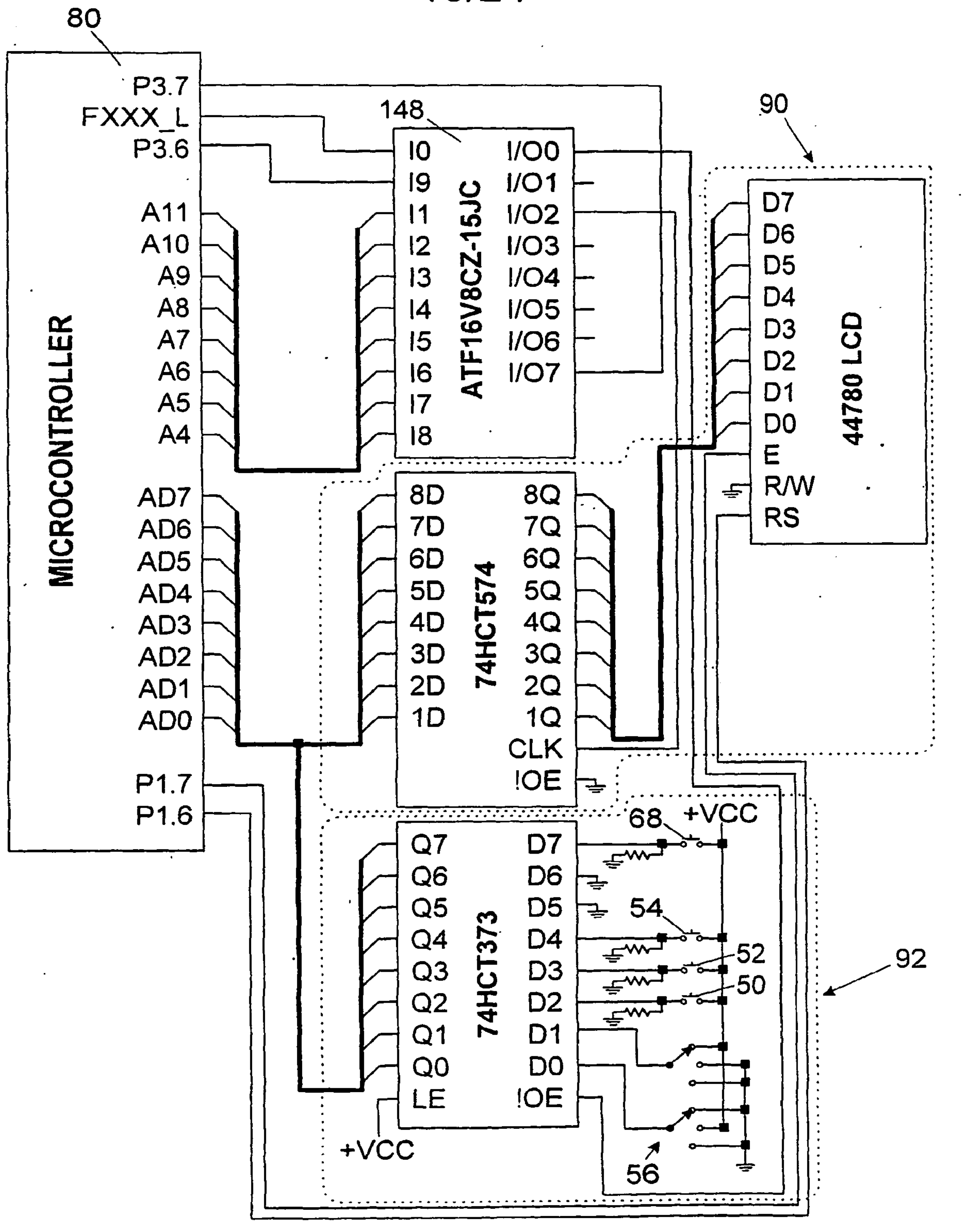

80
MICROCONTROLLER
P3.7
FXXX_L
P3.6
A11
A10
A9
A8
A7
A6
A5
A4
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
P1.7
P1.6
148
ATF16V8CZ-15JC
I0
I9
I1
I2
I3
I4
I5
I6
I7
I8
I/O0
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
74HCT574
8D
7D
6D
5D
4D
3D
2D
1D
8Q
7Q
6Q
5Q
4Q
3Q
2Q
1Q
CLK
!OE
74HCT373
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
LE
D7
D6
D5
D4
D3
D2
D1
D0
!OE
+VCC
90
44780 LCD
D7
D6
D5
D4
D3
D2
D1
D0
E
R/W
RS
68
+VCC
54
52
50
56
92

Fig. 11

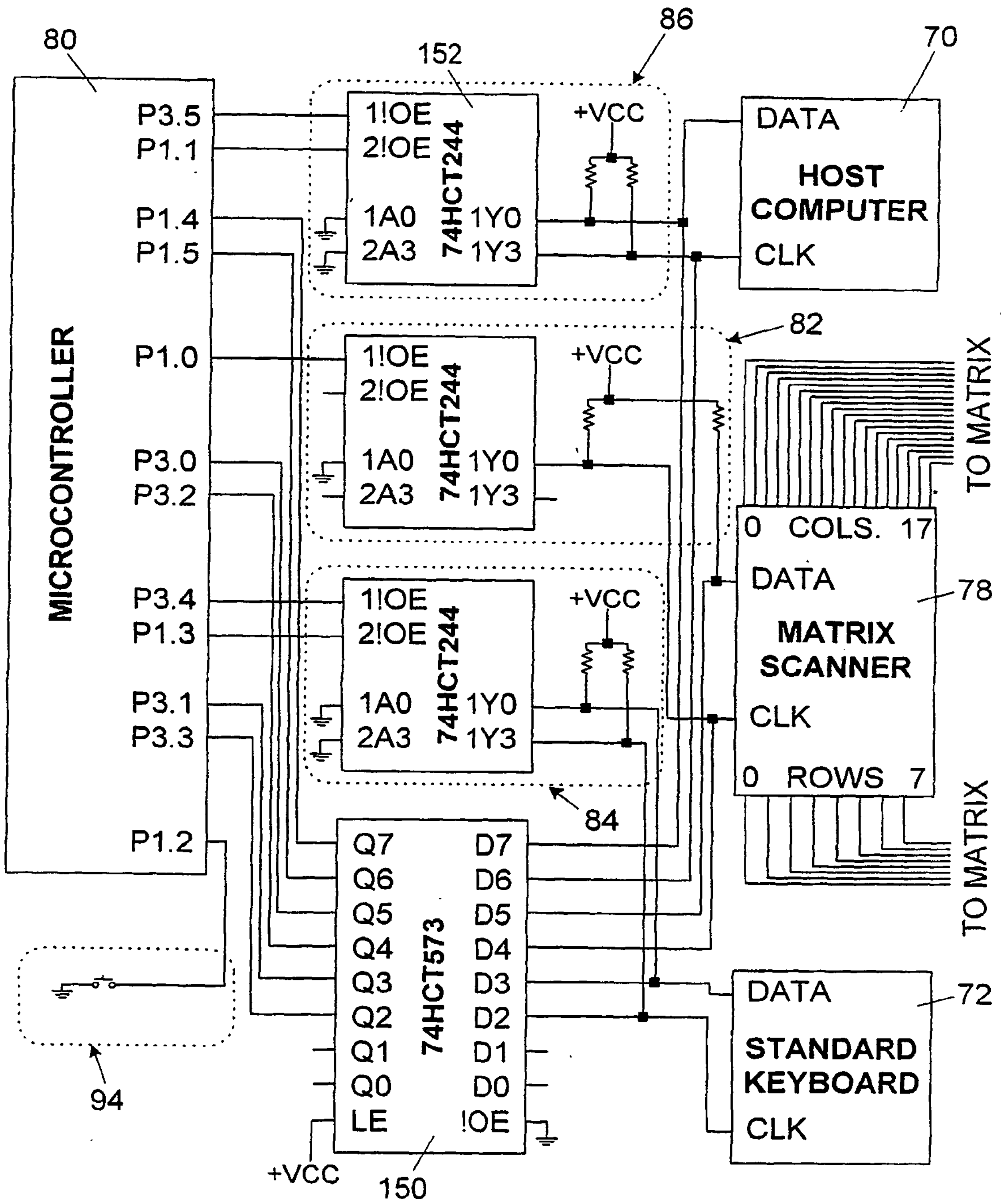

80
86
152
70
MICROCONTROLLER
P3.5
P1.1
P1.4
P1.5
P1.0
P3.0
P3.2
P3.4
P1.3
P3.1
P3.3
P1.2
1!OE
2!OE
1A0
2A3
74HCT244
1Y0
1Y3
+VCC
DATA
HOST COMPUTER
CLK
82
TO MATRIX
0 COLS. 17
DATA
MATRIX SCANNER
CLK
78
0 ROWS 7
84
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
LE
D7
D6
D5
D4
D3
D2
D1
D0
!OE
74HCT573
150
DATA
STANDARD KEYBOARD
CLK
72
94

Fig. 12

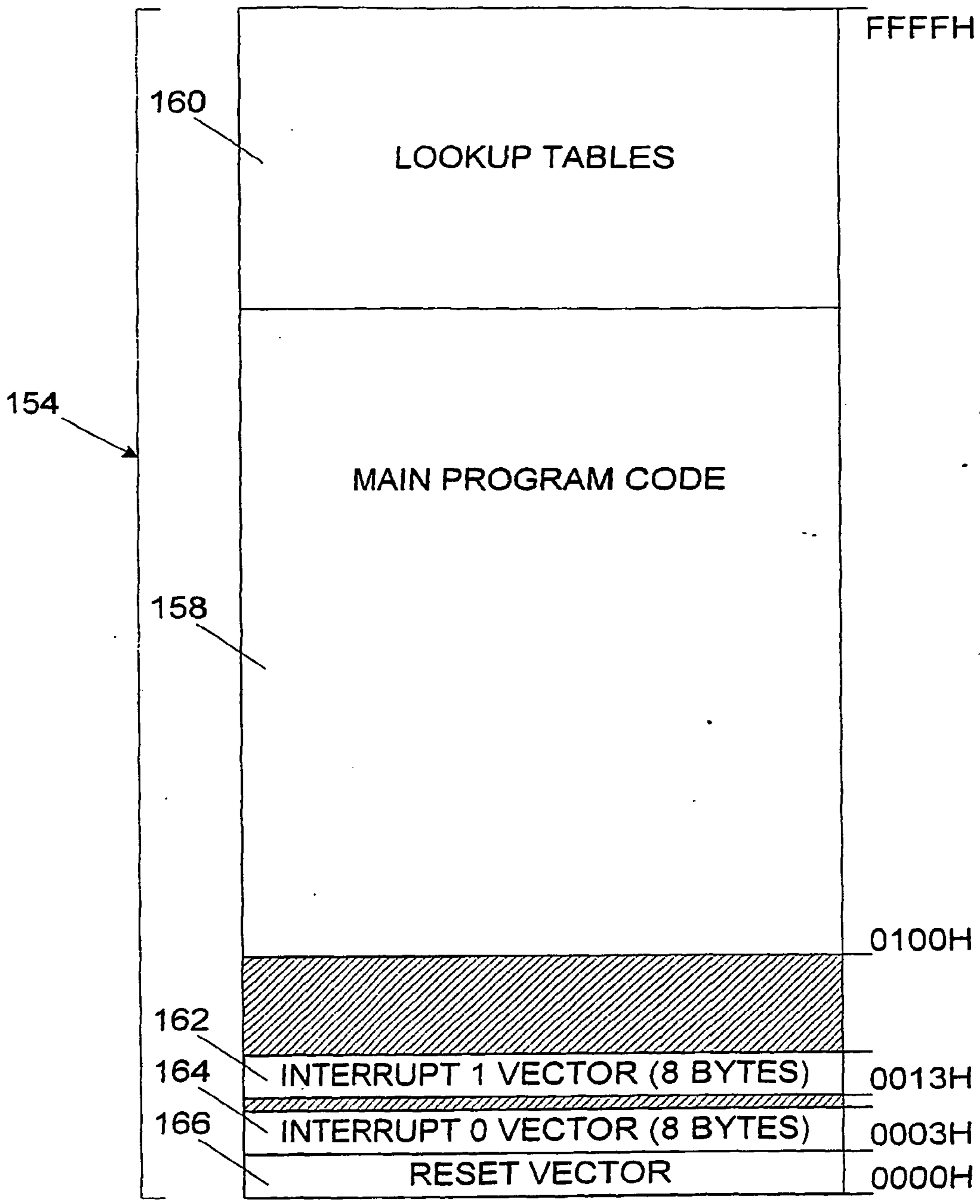
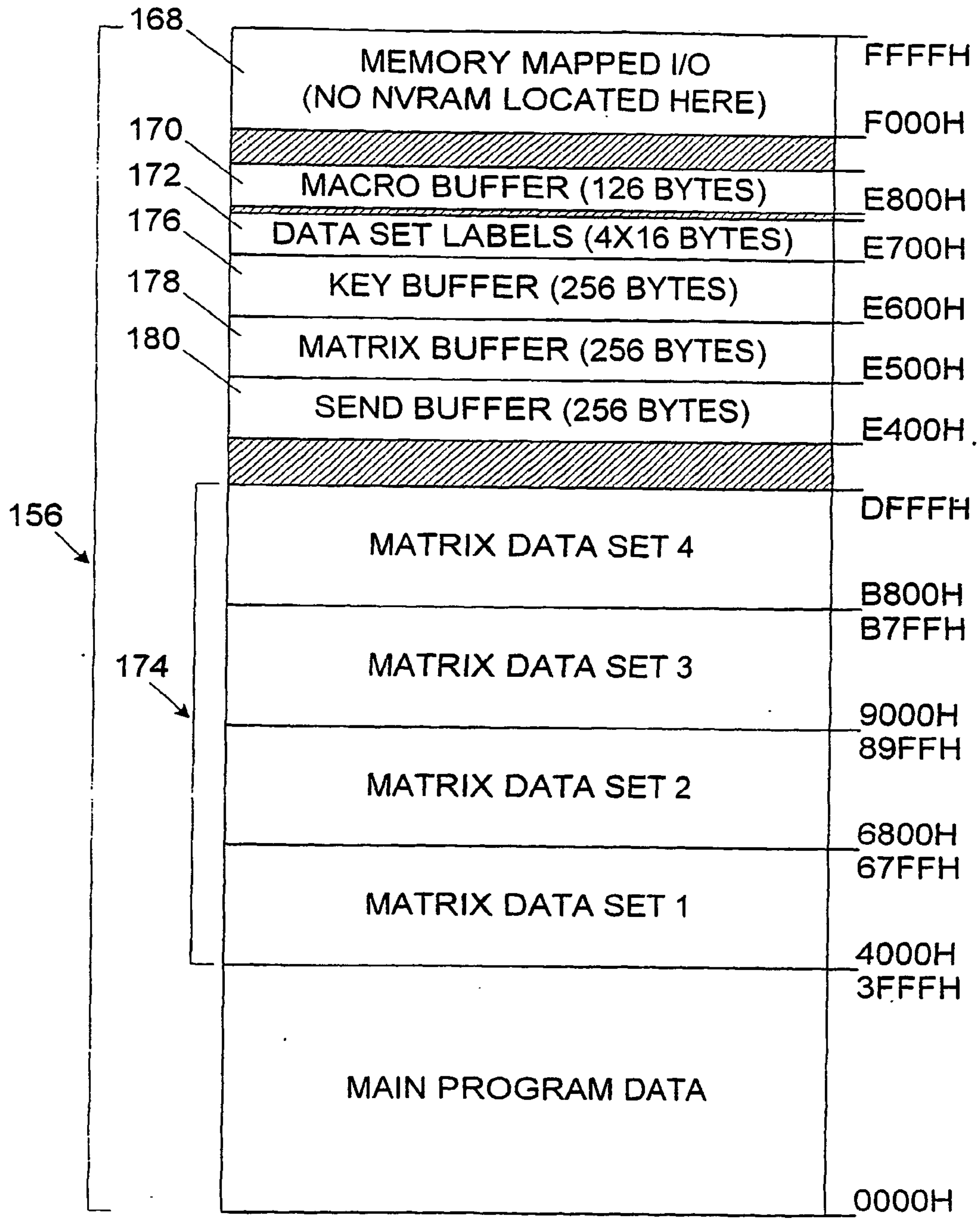

FFFFH
160
LOOKUP TABLES
154
MAIN PROGRAM CODE
158
0100H
162
INTERRUPT 1 VECTOR (8 BYTES)
0013H
164
INTERRUPT 0 VECTOR (8 BYTES)
0003H
166
RESET VECTOR
0000H

Fig. 13



168
MEMORY MAPPED I/O (NO NVRAM LOCATED HERE)
FFFFH
F000H
170
MACRO BUFFER (126 BYTES)
E800H
172
DATA SET LABELS (4X16 BYTES)
E700H
176
KEY BUFFER (256 BYTES)
E600H
178
MATRIX BUFFER (256 BYTES)
E500H
180
SEND BUFFER (256 BYTES)
E400H
156
DFFFH
MATRIX DATA SET 4
B800H
B7FFH
174
MATRIX DATA SET 3
9000H
89FFH
MATRIX DATA SET 2
6800H
67FFH
MATRIX DATA SET 1
4000H
3FFFH
MAIN PROGRAM DATA
0000H

Fig. 14

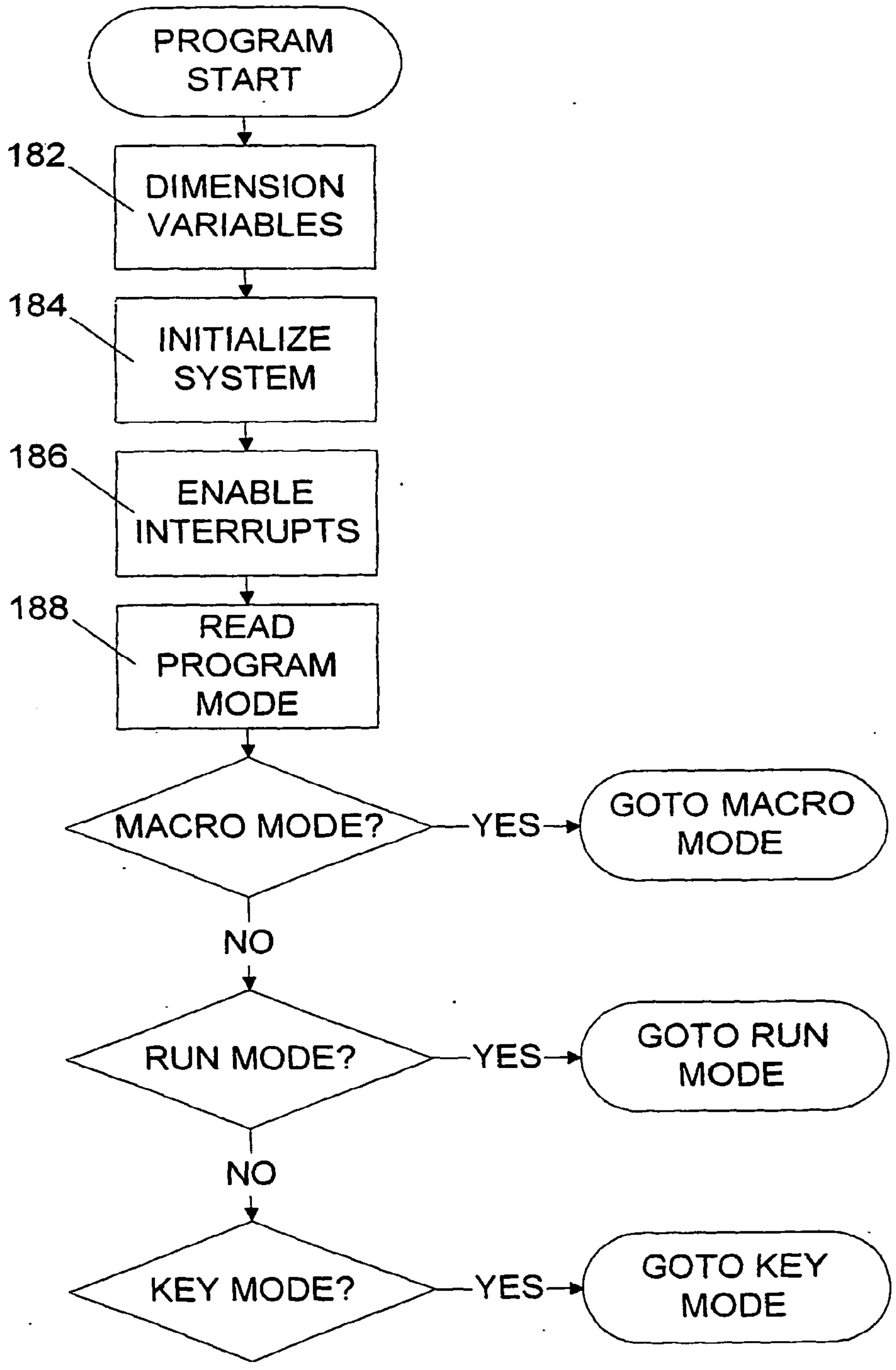

PROGRAM START
182
DIMENSION VARIABLES
184
INITIALIZE SYSTEM
186
ENABLE INTERRUPTS
188
READ PROGRAM MODE
MACRO MODE?
YES
GOTO MACRO MODE
NO
RUN MODE?
YES
GOTO RUN MODE
NO
KEY MODE?
YES
GOTO KEY MODE

Fig. 15

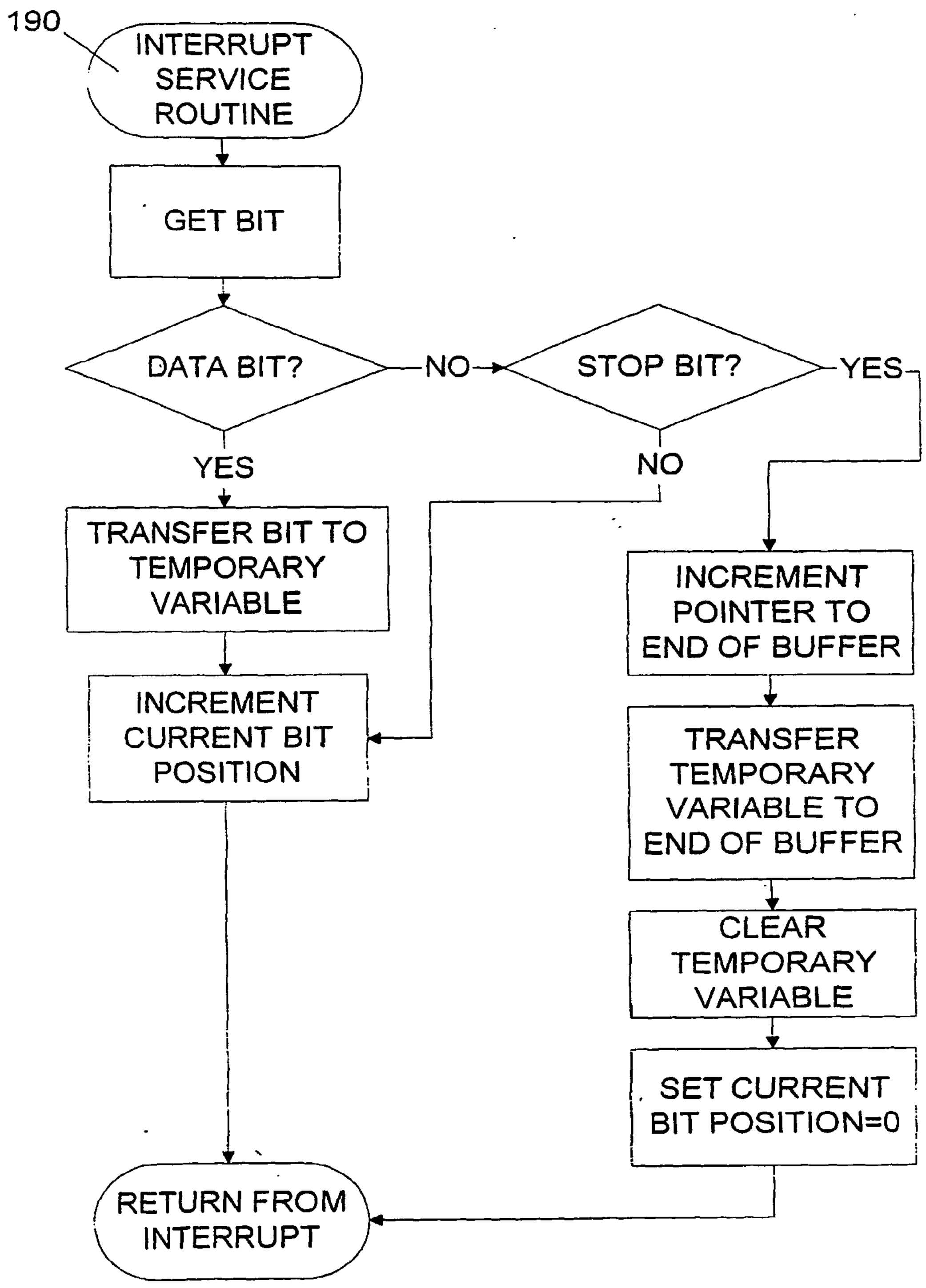

190
INTERRUPT SERVICE ROUTINE
GET BIT
DATA BIT?
NO
STOP BIT?
YES
YES
NO
TRANSFER BIT TO TEMPORARY VARIABLE
INCREMENT CURRENT BIT POSITION
INCREMENT POINTER TO END OF BUFFER
TRANSFER TEMPORARY VARIABLE TO END OF BUFFER
CLEAR TEMPORARY VARIABLE
SET CURRENT BIT POSITION=0
RETURN FROM INTERRUPT

Fig. 16

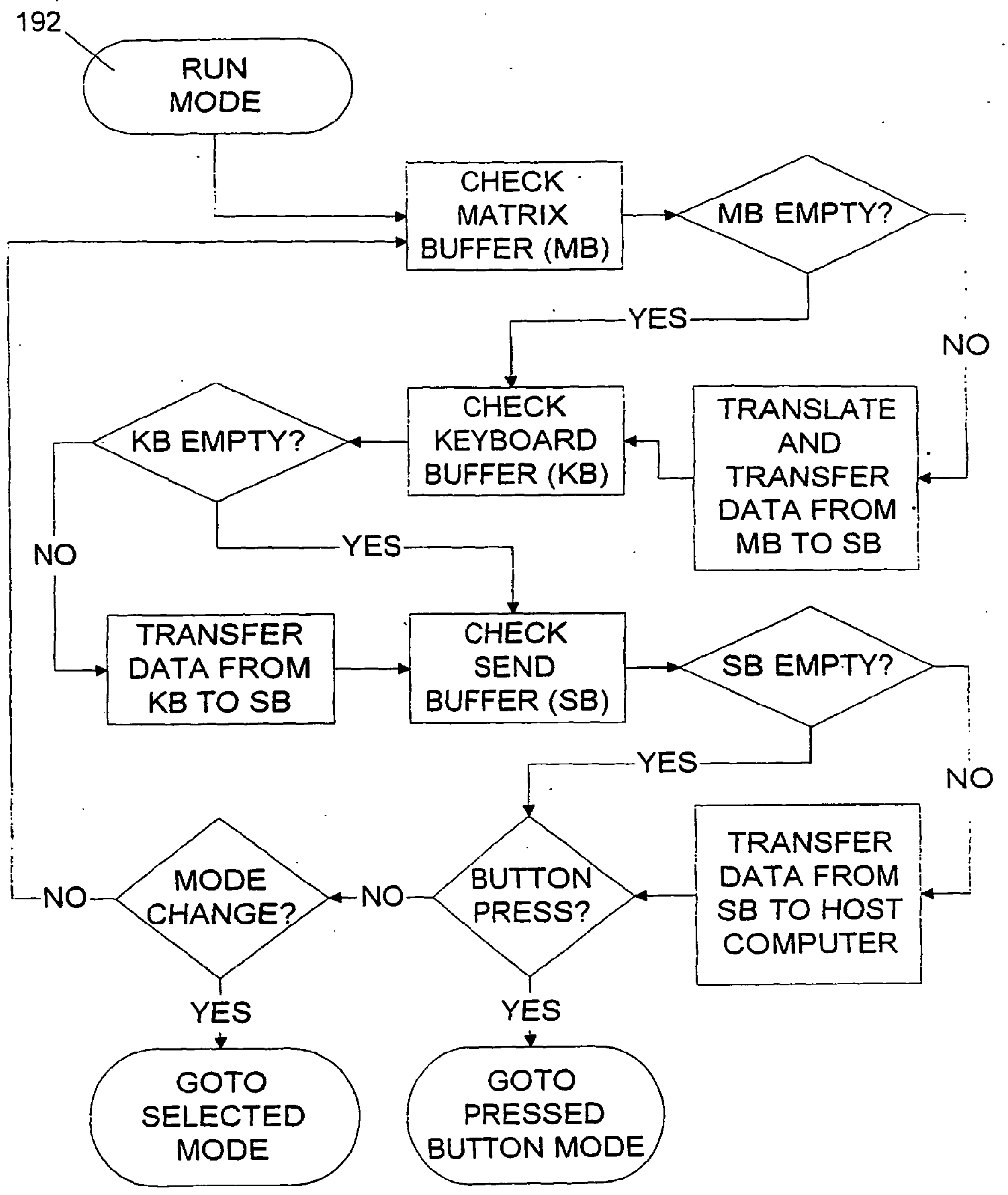

192
RUN MODE
CHECK MATRIX BUFFER (MB)
MB EMPTY?
YES
NO
CHECK KEYBOARD BUFFER (KB)
TRANSLATE AND TRANSFER DATA FROM MB TO SB
KB EMPTY?
NO
YES
TRANSFER DATA FROM KB TO SB
CHECK SEND BUFFER (SB)
SB EMPTY?
YES
NO
TRANSFER DATA FROM SB TO HOST COMPUTER
BUTTON PRESS?
NO
MODE CHANGE?
NO
YES
YES
GOTO SELECTED MODE
GOTO PRESSED BUTTON MODE

Fig. 17

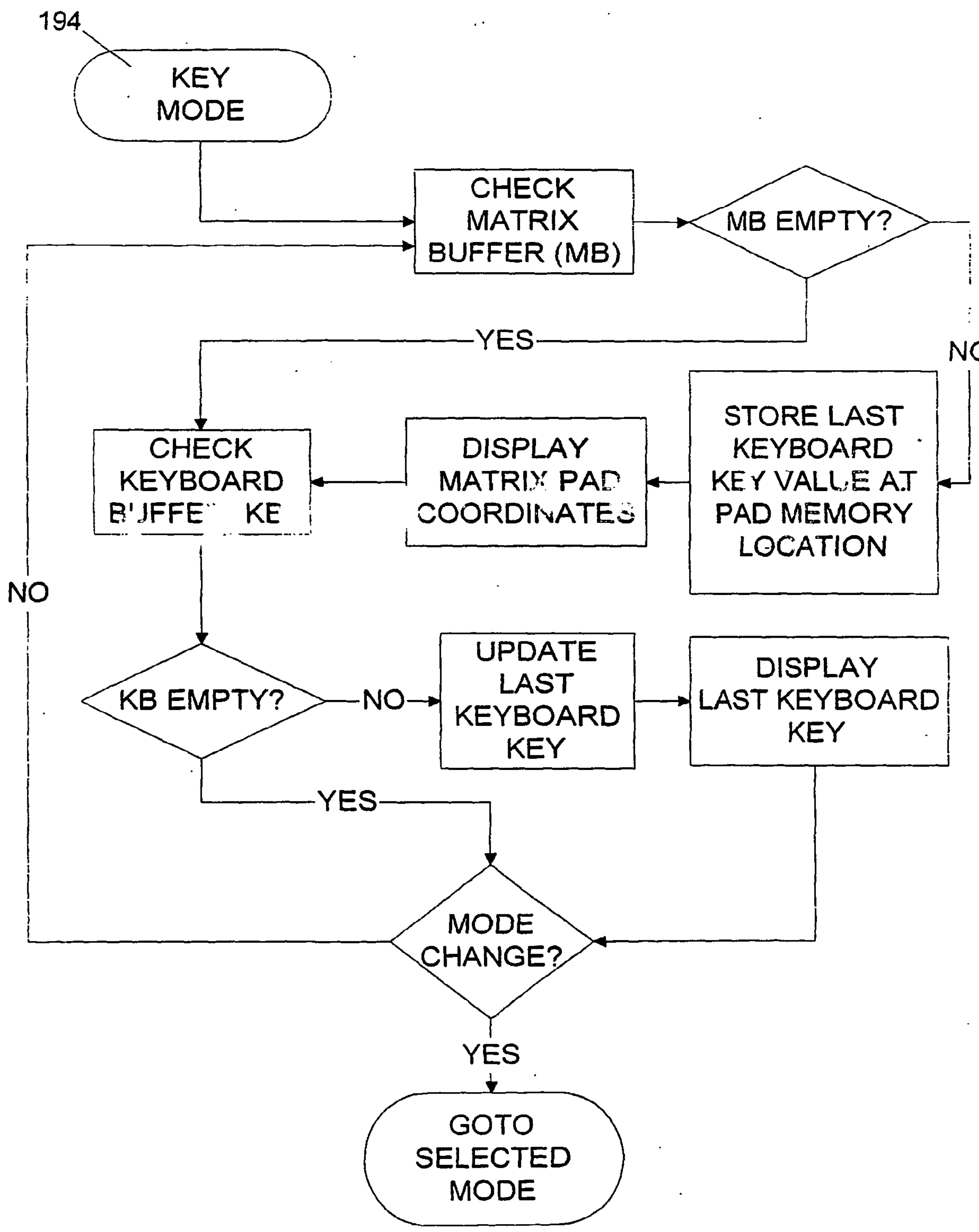

194
KEY MODE
CHECK MATRIX BUFFER (MB)
MB EMPTY?
YES
NO
CHECK KEYBOARD BUFFER KB
DISPLAY MATRIX PAD COORDINATES
STORE LAST KEYBOARD KEY VALUE AT PAD MEMORY LOCATION
NO
KB EMPTY?
NO
UPDATE LAST KEYBOARD KEY
DISPLAY LAST KEYBOARD KEY
YES
MODE CHANGE?
YES
GOTO SELECTED MODE

Fig. 18

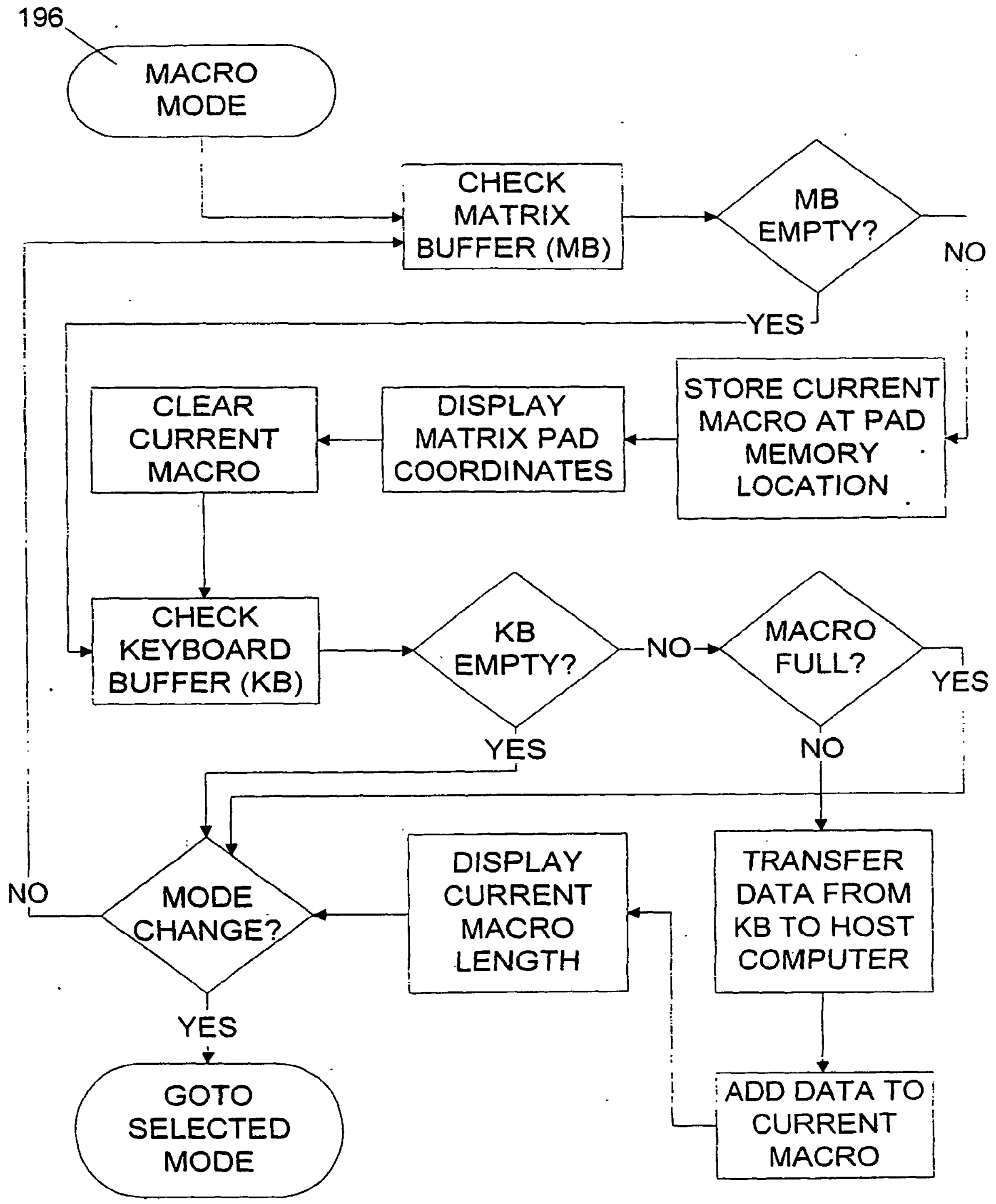

196
MACRO MODE
CHECK MATRIX BUFFER (MB)
MB EMPTY?
NO
YES
STORE CURRENT MACRO AT PAD MEMORY LOCATION
DISPLAY MATRIX PAD COORDINATES
CLEAR CURRENT MACRO
CHECK KEYBOARD BUFFER (KB)
KB EMPTY?
NO
MACRO FULL?
YES
YES
NO
NO
MODE CHANGE?
DISPLAY CURRENT MACRO LENGTH
TRANSFER DATA FROM KB TO HOST COMPUTER
YES
GOTO SELECTED MODE
ADD DATA TO CURRENT MACRO

Fig. 19

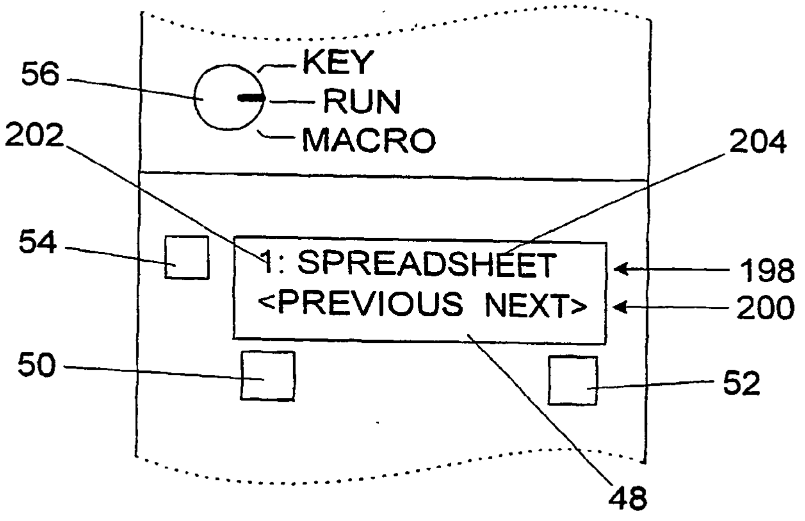


Fig. 20A

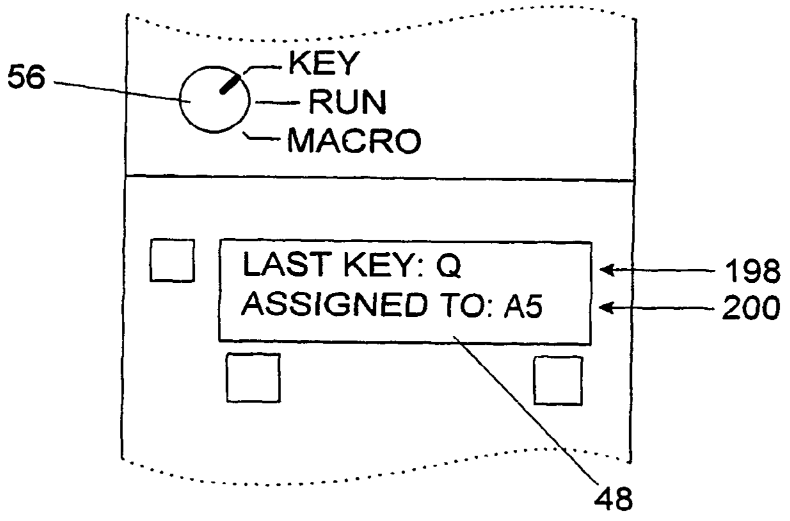


Fig. 20B

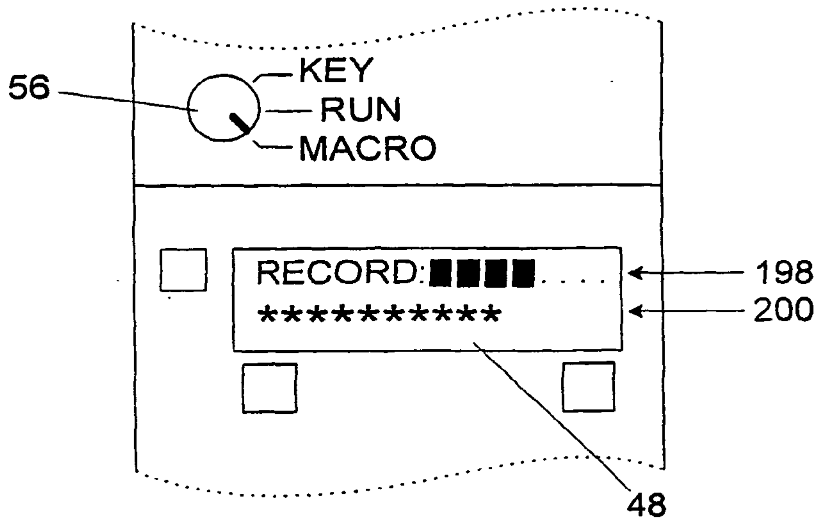


Fig. 20C

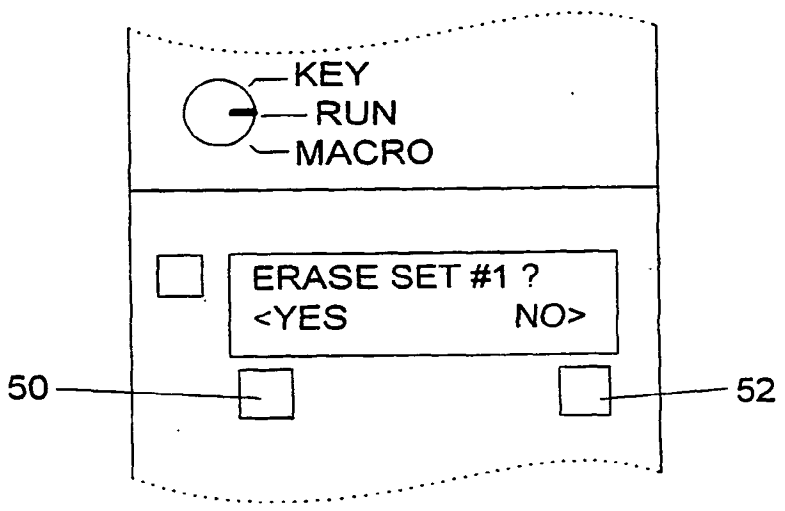


Fig. 20D

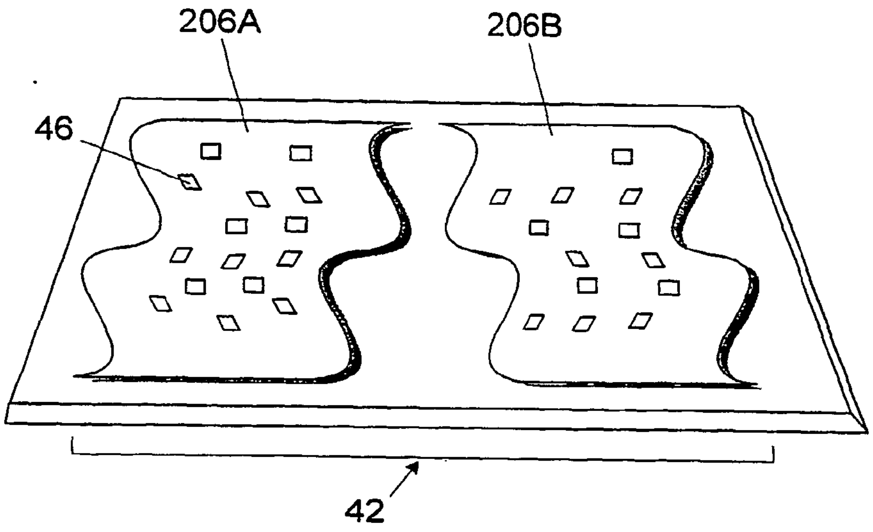


Fig. 21

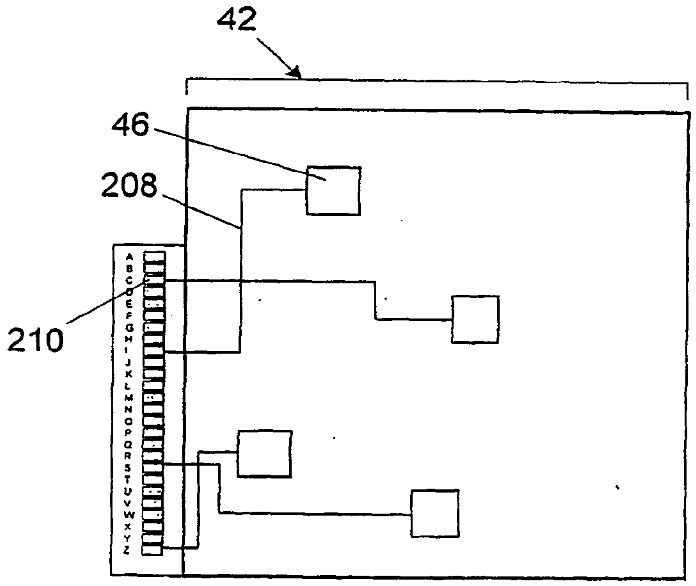


Fig. 22

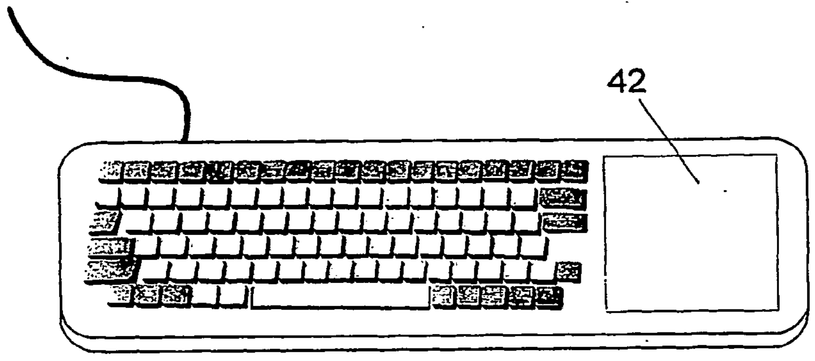


Fig. 23

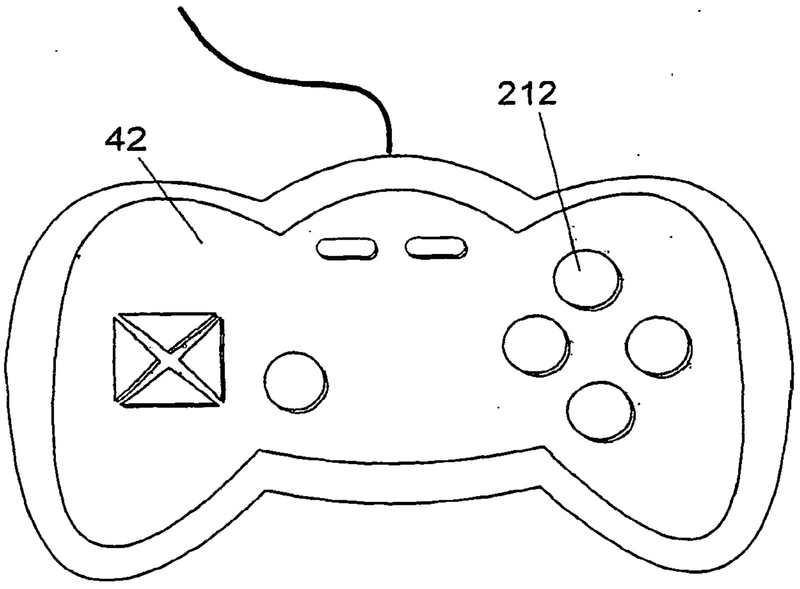


Fig. 24

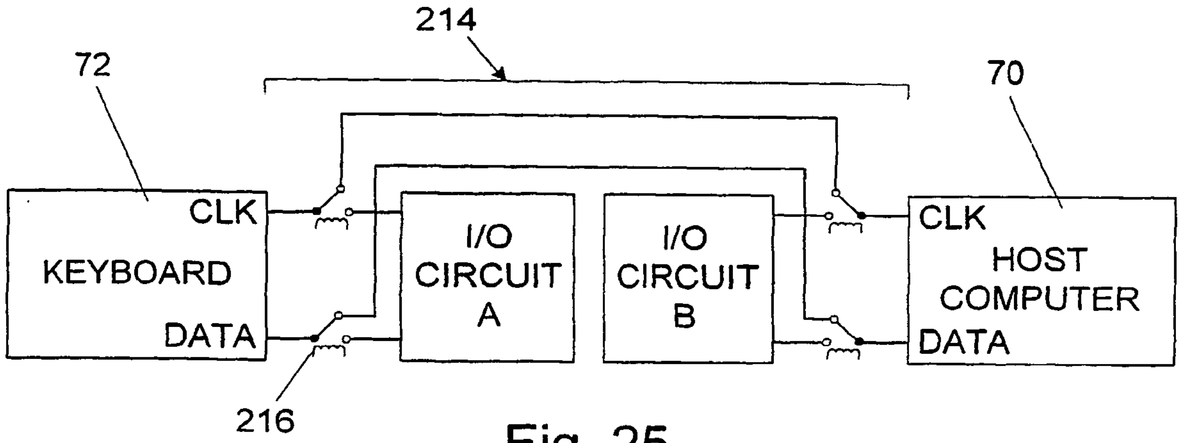

214
72
70
KEYBOARD
CLK
DATA
I/O CIRCUIT A
I/O CIRCUIT B
CLK
HOST COMPUTER
DATA
216

Fig. 25

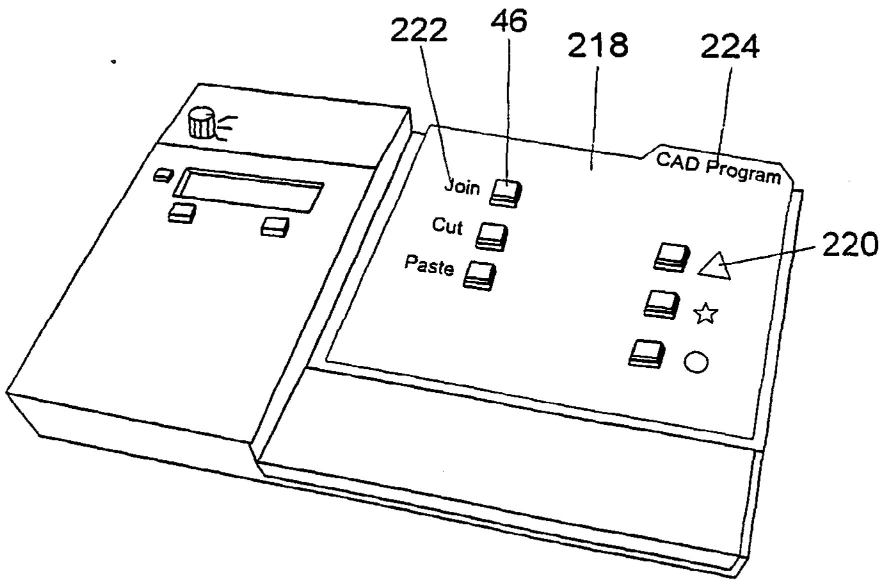

222
46
218
224
CAD Program
Join
Cut
Paste
220

Fig. 26

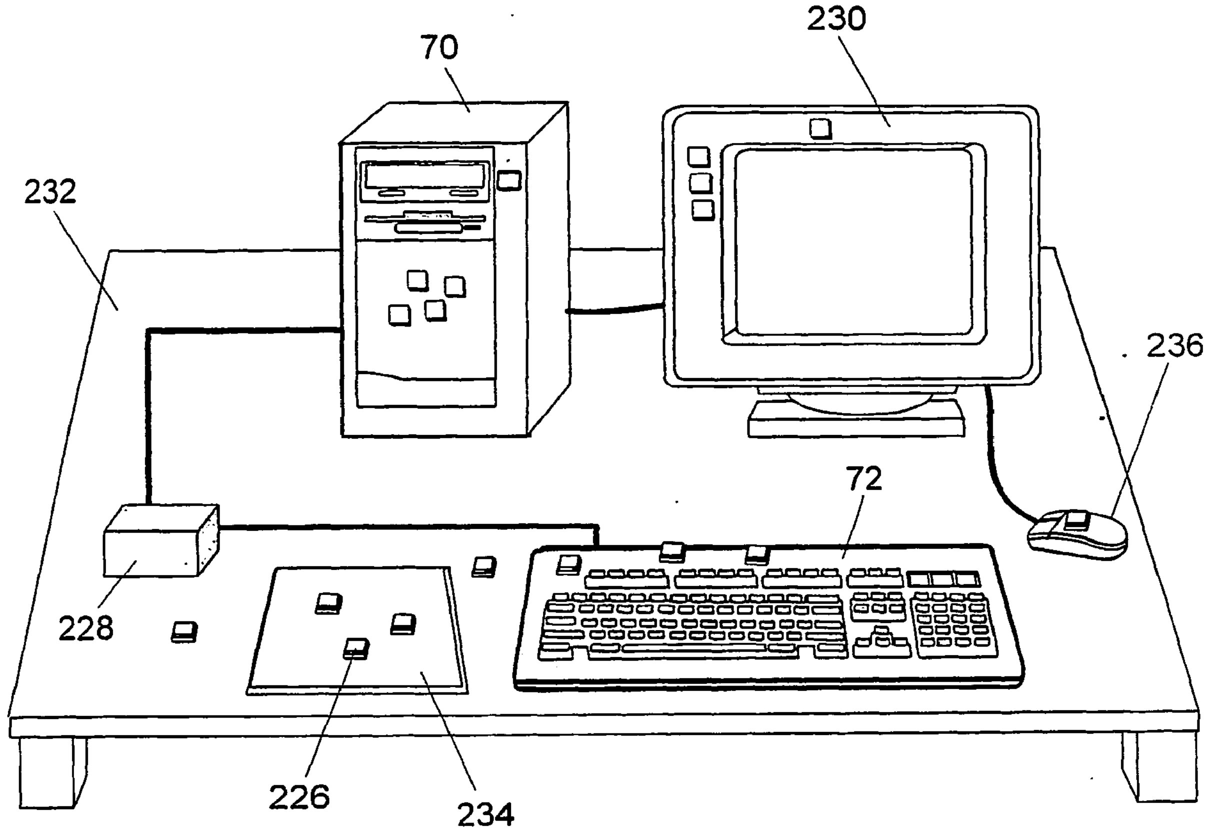


Fig. 27

```
;Source Code for "Title of Patent Application"
;Written by Scott M. Rix
;Submitted for-Patent on "Date"

;///////////////////////////////////////////////////////////////////////////////
;The following software source code was designed to operate on the
;33 Mhz Dallas 80C320 HSM/KISS Development System sold by:
;
;Systronix(R), Inc.,
;555 South 300 East #21
;Salt Lake City, Utah  84111
;
;The source code is a hybrid of BASIC and Assembly language.
;BASIC is used for the overall program flow control and string manipulation
;while in-line Assembly language is used for speed sensitive
;operations.
;This source code was compiled using the Systronix(R), Inc. BCI51(TM) PRO BASIC
;Cross Compiler, Version 1.40
;///////////////////////////////////////////////////////////////////////////////

;///////////////////////////////////////////////////////////////////////////////
;**Compiler****Compiler****Compiler****Compiler****Compiler****Compiler**
;///////////////////////////////////////////////////////////////////////////////
;                          Compiler Directives

#TARGET 80C32                   ;Uses the Dallas 80C320 Processor
#XTAL  33000000                 ;The Crystal Frequency is 33.0000 MHz
#CONSOLE MODE=19200             ;Console Comm is at 19.2 kbaud
#ISTACK START = 042H            ;Start of Stack Memory is 042H
#CODE START 0100H               ;Start of Code Memory is 0100H
#DATA START 0H                  ;Start of Data Memory is 00H
#CHECK MATH OFF                 ;Compiler directive to ignore variable type
                                ;compatibility tests
#INITIALIZE VARIABLES OFF       ;Save Data memory between program runs
```

```
;///////////////////////////////////////////////////////////////////////////////
;**BASIC Variables****BASIC Variables****BASIC Variables****BASIC Variables**
;///////////////////////////////////////////////////////////////////////////////
;                          Basic Variable Definition

;***Counting variables for main routine
UNSIGNED INTEGER i              ;counting/temp variable
UNSIGNED CHARACTER j            ;counting/temp variable
UNSIGNED CHARACTER k            ;counting/temp variable
UNSIGNED INTEGER l              ;counting/temp variable
UNSIGNED INTEGER m              ;counting/temp variable

;***Counting variables for subroutines
UNSIGNED INTEGER ii             ;counting/temp variable
UNSIGNED CHARACTER jj           ;counting/temp variable
UNSIGNED CHARACTER kk           ;counting/temp variable

UNSIGNED INTEGER intTemp        ;temporary variable
UNSIGNED CHARACTER chaTemp      ;temporary variable
STRING strTemp$(1)              ;temporary variable

;***Delay variables
UNSIGNED CHARACTER msec_delay_value       ;holds the number of milliseconds
                                          ;(integer 0-255) needed in the
                                          ;Msec_Delay subroutine

;***LCD String Variables
STRING STRtoLCD1$(16)           ;a 16-byte string that holds the contents
                                ;to be displayed on the entire first
                                ;line of the LCD.  Passed to the Subroutine
                                ;String_To_LCD1

STRING STRtoLCD2$(16)           ;a 16-byte string that holds the contents
                                ;to be displayed on the entire second
                                ;line of the LCD.  Passed to the Subroutine
                                ;String_To_LCD2

UNSIGNED INTEGER PM_Start1      ;The starting location in memory to Print Out on
                                ;line 1 of LCD
UNSIGNED INTEGER PM_Start2      ;The starting location in memory to Print Out on
                                ;line 2 of LCD


;***Program Control Variables
UNSIGNED CHARACTER Prog_Mode    ;Holds the current program mode
                                ;0=Macro, 1=Run, 2=Key

UNSIGNED CHARACTER Button_Delay      ;delay for the Left, Right, Clear, and
                                     ;Rename Key debounce

UNSIGNED INTEGER EIE            ;EIE = Extended Interrupt Enable Register
                                ;Stores the location of the EIE Register
```

```
;***Data Set Variables
STRING DataSet1$(16)        ;LCD Label for Data set 1
STRING DataSet2$(16)        ;LCD Label for Data set 2
STRING DataSet3$(16)        ;LCD Label for Data set 3
STRING DataSet4$(16)        ;LCD Label for Data set 4

STRING DisplayLabel$(16)    ;Label building string for Data sets
                            ;used during the Label routine


;***** Matrix Programming Variables
STRING Matrix_Position$(2)          ;Holds two-letter row-column designator
                                    ;string for a Key in the Matrix.

UNSIGNED CHARACTER B_Matrix_Code    ;the BASIC variable that gets the Assembly
                                    ;variable Matrix_Code

STRING Key_Label$(7)                ;Holds the text description of the Keyboard
                                    ;Key that is pressed during Key Mode.

UNSIGNED CHARACTER B_Key_Code       ;the BASIC variable that gets the Assembly
                                    ;variable Key_Code

UNSIGNED CHARACTER B_E0_Flag        ;the BASIC variable that gets the Assembly
                                    ;variable E0_Flag

UNSIGNED CHARACTER B_AB_Length      ;the BASIC variable that gets the Assembly
                                    ;variable AB_Length

UNSIGNED INTEGER Num_Bars           ;The number of Bars to display on the
                                    ;LCD line 1 during Macro Recording

UNSIGNED INTEGER Num_Asterisks      ;The number of Asterisks to display on the
                                    ;LCD line 2 during Macro Recording

;***********************************************************

;///////////////////////////////////////////////////////////////////////////////
;**Main****Main****Main****Main****Main****Main****Main****Main****Main****Main*
;///////////////////////////////////////////////////////////////////////////////

Main:

;**** Define Basic Constants
EIE=0E8H     ;Define the location of the Extended Interrupt Enable Register
```

```
;/////////////////////////////////////////////////////////////////////////////
;                          Assembly Variable Definition
;/////////////////////////////////////////////////////////////////////////////
#ASM
;///////// HSM Pin Input/Output /////////

;**** Matrix Bits
Matrixi_CLK BIT P3.2       ;This bit is input only. It is connected
                           ;to the INT0 pin.  When enabled, ISR0
                           ;is triggered on a falling edge.

Matrixi_Data BIT P3.0      ;This bit is input only.  It is used to read the
                           ;status of the Matrix data line during ISR0.

OMatrixo_CLK BIT P1.0      ;This bit is output only. Used to drive the CLK Line
                           ;to the Matrix.


;**** Keyboard Bits
Keyi_CLK BIT P3.3          ;This bit is input only.  It is connected to the
                           ;INT1 pin.  When enabled, ISR1 is triggered on a
                           ;low falling edge.

Keyi_Data BIT P3.1         ;This bit is input only.  It reads the status of
                           ;the data line during ISR1 to get the Keyboard
                           ;information.

OKeyo_CLK BIT P1.3         ;Output only. Used to drive the CLK line to the
;Keyboard.

OKeyo_Data BIT P3.4        ;Output only. Used to drive the Data line to the
;Keyboard.


;**** Clock Bits
CPUi_CLK BIT P1.5          ;This bit is input only.

CPUi_Data BIT P1.4         ;This bit is input only.  It reads the status of
                           ;the CPU data line

OCPUo_CLK BIT P1.1         ;Output only. Used to drive the CLK line to the CPU.

OCPUo_Data BIT P3.5        ;Output only. Used to drive the Data line to the CPU.


;**** Matrix Data Transfer variables
Matrix_Bit_Num EQU 030H          ;the current bit received for the curent byte
                                 ;from the Matrix
                                 ;holds a value between 0 and 10
                                 ;0 = start bit
                                 ;1..8 = data bits
                                 ;9 = parity bit
                                 ;10 = stop bit
```

```
Matrix_Byte EQU 031H     ;the current byte being received from the Matrix
                         ;built up one bit at a time

Matrix_Bit BIT 00H       ;the value of the current bit (0 or 1) received from
                         ;the Matrix


Matrix_Sending BIT 08H   ;Flag to indicate the status of the Matrix
                         ;0=idle, 1=Sending data


;**** Keyboard data transfer variables
Key_Bit_Num EQU 036H     ;the current bit received for the current byte
                         ;from the Matrix
                         ;holds a value between 0 and 10
                         ;0 = start bit
                         ;1..8 = data bits
                         ;9 = parity bit
                         ;10 = stop bit

Key_Byte EQU 037H ;the current byte being received from the Matrix
                  ;built up one bit at a time

Key_Bit BIT 03H   ;the value of the current bit (0 or 1) received from
                  ;the Matrix

Key_Sending BIT 0AH      ;Flag to indicate the status of the Keyboard
                         ;0=idle, 1=Sending data


;**** CPU Data Transfer variables
CPU_Bit_Num EQU 37H      ;The current bit sent to the Keyboard from the CPU
                         ;1..8 = data bits
                         ;9 = parity bit

CPU_Sending BIT 09H      ;Flag to indicate the status of the CPU
                         ;0=idle, 1=Sending data


;**** Send Buffer Variables
;The Send Buffer is a 256 byte circular Buffer located in external memory
;That holds a sequential string of bytes that need to be sent to the CPU
;The run mode loop waits until there is no traffic between the CPU and/or
;the Keyboard before Sending the bytes from the Buffer

SB_Current EQU 03BH      ;0-255 the position of the current byte in the circular
;Buffer
SB_Last EQU 03CH  ;0-255 , the position of the last byte in the ;circular
;Buffer
SB_DPH EQU 03DH          ;The high byte pointing to the circular Send Buffer in
;external memory
                         ;used to fill the DPH before a MOVX
```

```
Traffic_Delay EQU 03EH  ;Holds a delay that gets reset every time an INT1 or
;INT3 is executed
;the delay is used to make sure the Matrix does not step
;on traffic between the CPU and Keyboard.  The delay is
;greater than the time between clock pulses, to make
;sure any transmissions are completed before the
;Send Buffer Sends a byte


;**** Matrix Buffer Variables
;The Matrix Buffer is a 256 byte circular Buffer located in external memory
;that holds a sequential string of bytes that has been received from the Matrix
;This Buffer is used to  prevent ISR0 from taking too long.  In ISR0, bytes
;are added to the Matrix Buffer in the order they are received.  The main
;program then uses the receive Buffer to get the information from thee lookup
;table and Matrix the appropriate data to the Send Buffer.

MB_Current EQU 023H      ;0-255 the position of the current byte in the circular
;Buffer
MB_Last EQU 024H         ;0-255 , the position of the last byte in the circular
;Buffer
MB_DPH EQU 025H          ;The high byte pointing to the circular Matrix Buffer in
;external memory
                         ;used to fill the DPH before a MOVX
MB_Release BIT 01H       ;Flag to detect if the detected Matrix Key was
;pressed=0, or released=1


;**** Keyboard Buffer Variables
;The Keyboard Buffer is a 256 byte circular Buffer located in external memory
;that holds a sequential string of bytes that has been received from the
;Keyboard. This Buffer is used to prevent ISR1 from taking too long.  In
;ISR1, bytes are added to the Keyboard Buffer in the order they are received.
;The main program then transfers these bytes to the Send Buffer

KB_Current EQU 026H      ;0-255 the position of the current byte in the circular
                         ;Buffer
KB_Last EQU 027H         ;0-255 , the position of the last byte in the circular
                         ;Buffer
KB_DPH EQU 028H          ;The high byte pointing to the circular Matrix Buffer in
;external memory used to fill the DPH before a MOVX

;**** Macro Buffer Variables
;The Macro is a 126 byte Buffer located in external memory
;that holds a sequential string of bytes received from the Keyboard
;This Buffer is filled during Macro programming mode as bytes are transferred
;from the Send Buffer to the CPU.  When the user presses a Matrix Key, the byte
;stream recorded in the Macro Buffer are transferred to the pad location
;specified by the Matrix Key location and the current data set.  (Note: the
;current design and memory allocation plan allows a maximum of 126 bytes to be
;recorded for each Macro.  Each pad has 128 bytes reserved - 1st byte to
;indicate Macro - 2nd byte to indicate Macro length =126)

AB_DPH EQU 01FH          ;The high byte pointing to the circular Macro Buffer in
                         ;external memory
                         ;used to fill the DPH before a MOVX
AB_Length EQU 01EH       ;the number of bytes recorded in the current Macro
```

```
;*** General Bits
Send_Parity BIT 02H       ;The parity bit for a byte sent to the CPU or Keyboard

CAPS_Down Bit 0CH         ;Flag to indicate Caps lock status, 1=pressed,
                          ;0=released

NUM_Down Bit 0DH          ;Flag to indicate Num lock status, 1=pressed, 0=released

SCROLL_Down Bit 0EH       ;Flag to indicate scroll lock status, 1=pressed,
                          ;0=released

SB_Prev_Byte EQU 3AH      ;Holds the previous byte sent from the SB
                          ;used to check for F0 (keey release)

System_Reset Bit P2.1     ;The System reset pin is used to initialize the
                          ;system after a new program is downloaded.
                          ;it is not accessible to the user and is triggered by
                          ;a HSMK board mounted push button.

;*** Collision Check
Prev_Key_Num EQU 029H             ;The Key bit number on the last check
Key_Checks_Left EQU 02AH          ;The number of checks left before a collision
                                  ;is signaled

Prev_Matrix_Num EQU 02BH          ;The Matrix bit number on the last check
Matrix_Checks_Left EQU 02CH       ;The number of checks left before a collision
                                  ;is signaled.

;*** Data Set variable
Data_Set EQU 02EH         ;Hold the memory location offset (DPH) for the currently
                          ;active data set
                          ;For set 1=0H, set 2=28H, set 3=50H, set 4=78H
                          ;used to calculate the lookup table pointer to the data
                          ;bound to the Matrix Pads.

;**** Key assign and Display label variables for Data Sets 1-4
Flag_E0 BIT 016H          ;Flag to check if E0 was included with Keyboard press
                          ;detected during label and assign Key functions

Flag_F0 BIT 017H          ;Flag to check if F0 was included with Keyboard press
                          ;detected during label and assign Key functions

Flag_M_F0 BIT 015H        ;Flag to check if F0 was included with Matrix code
                          ;detected during assign Key function

Pass_Key_Value EQU 040H   ;used to pass Key code value from BASIC to Assembly
Pass_E0_Value EQU 041H    ;used to pass E0 value from BASIC to Assembly

Matrix_Code EQU 02DH      ;The last key code received from the matrix
Key_Code EQU 02FH         ;The last key code received from the keyboard

LBL_PNTR EQU 033H         ;1-16, points to the current position in the label
LBL_Offset EQU 032H       ;0, 010H, 020H, 030H corresponding to the current data
                          ;set.
#ASM_END
```

```
;///////////////////////////////////////////////////////////////////////////////
;**Main****Main****Main****Main****Main****Main****Main****Main****Main****Main*
;///////////////////////////////////////////////////////////////////////////////
;                          Main Startup

Main_Startup:
#ASM
;*** Set input only bits HI for input
      ;(Setting the input bits HI prepares them for data input)
      SETB Matrixi_CLK         ;Set the INT0/Matrix CLK pin for input
      SETB Matrixi_Data        ;Set the Matrix Data pin for input.

      SETB Keyi_CLK            ;Set the Keyboard CLK input
      SETB Keyi_Data           ;Set the Keyboard Data input

      SETB CPUi_CLK            ;Set the CPU CLK input
      SETB CPUi_Data           ;Set the CPU Data input


      ;Set output bits HI for normal operation
      ;(Setting the output bits puts the connected line driver
      ;into a high impedance (High Q) state and allows data
      ;to be passed to/from the specified line.)
      SETB OMatrixo_CLK

      SETB OKeyo_CLK
      SETB OKeyo_Data

      SETB OCPUo_CLK
      SETB OCPUo_Data

      MOV A,#0H                ;Put a zero in the Accumulator

      ;*** Clear Matrix Variables
      MOV Matrix_Bit_Num,A     ;Initialize the Matrix bit number
      MOV Matrix_Byte,A        ;Initialize the Matrix byte
      CLR Matrix_Bit           ;Initialize the current Matrix bit value
      CLR Matrix_Sending       ;Initialize the Matrix Sending flag

      ;*** Clear CPU Variables
      MOV CPU_Bit_Num,A        ;Initialize the CPU bit number
      CLR CPU_Sending          ;Initialize the CPU Sending flag

      ;*** Clear Keyboard Variables
      MOV Key_Bit_Num,A        ;Initialize the Key bit number
      MOV Key_Byte,A           ;Initialize the Key byte
      CLR Key_Bit              ;Initialize the current Key bit value
      CLR Key_Sending          ;Initialize the Key Sending flag
```

```
        ;Initialize Matrix, Key, and Send Buffer memory pointers
        MOV A,#0E4H        ;Send Buffer => the 256 bytes from E400H to E4FFH
        MOV SB_DPH,A

        MOV A,#0E5H        ;Matrix Buffer => the 256 bytes from E500H to E5FFH
        MOV MB_DPH,A


        MOV A,#0E6H        ;Key Buffer => the 256 bytes from E600H to E6FFH
        MOV KB_DPH,A

        MOV A,#0E8H        ;Macro Buffer => 126 bytes from E8700H to E87EH
        MOV AB_DPH,A


        ;Set the System_Reset pin HI
        ;System reset occurs when P1.2 is pulled LO
        ;during the Run Mode loop.  Access to system
        ;reset is only on the HSMK board using
        ;jumper JP9
        SETB System_Reset

#ASM_END

        ;**** Reset the LCD for 8-bit Communication
        GOSUB Clear_Display

        ;wait for the power-on self test to be completed by the
        ;Keyboard controller and the Matrix controller.
        STRtoLCD1$=" VTI Gutenboard " : GOSUB String_to_LCD1
        STRtoLCD2$="   Controller   " : GOSUB String_to_LCD2

        ;Create splash screen delay
        FOR i=1 TO 20000
              j=i
        NEXT i

        ;Initialize Send and Matrix Buffers
        FOR i=0C400H to 0C5FFH
              XBY(i)=00H
        NEXT i


;///////////////////////////////////////////////////////////////////////////
;**Main****Main****Main****Main****Main****Main****Main****Main****Main****Main*
;///////////////////////////////////////////////////////////////////////////
;                              Check Mode

        ;**** Get the current Matrix mode (run/Key/Macro)
        Prog_Mode=XBY(OFFC0H)          ;Program control bits are memory mapped at
                                       ;FFCXH.  Bit 0 controls the Run/Bind mode
                                       ;and Bit 1 controls the Key/Macro mode.

        Prog_Mode=Prog_Mode AND 0000_0011B
```

```
      IF Prog_Mode=0 THEN GOTO Macro_Mode
      IF Prog_Mode=1 THEN GOTO Run_Mode
      IF Prog_Mode=2 THEN GOTO Key_Mode


;////////////////////////////////////////////////////////////////////////////////
;**Run****Run****Run****Run****Run****Run****Run****Run****Run****Run****Run****
;////////////////////////////////////////////////////////////////////////////////
;                          Run Mode Startup

Run_Mode:

      ;Initialize the Matrix Buffer
#ASM
RM_Start:
      MOV A,#00H
      MOV MB_Current,A
      MOV MB_Last,A
      CLR MB_Release
#ASM_END

      ;Initialize the Key Buffer
#ASM
      MOV A,#00H
      MOV KB_Current,A
      MOV KB_Last,A
#ASM_END

      ;Initialize the Send Buffer
#ASM
      MOV A,#00H
      MOV SB_Current,A
      MOV SB_Last,A
      MOV SB_Prev_Byte,A

      CLR CAPS_Down
      CLR NUM_Down
      CLR SCROLL_Down
#ASM_END

      ;Initialize the Collision Check Variables
#ASM
      MOV A,#00H
      MOV Prev_Key_Num,A
      MOV Prev_Matrix_Num,A

      MOV A,#030H
      MOV Key_Checks_Left,A
      MOV Matrix_Checks_Left,A
#ASM_END

      ;Get the data labels
      GOSUB Get_lblDS1
      GOSUB Get_lblDS2
      GOSUB Get_lblDS3
      GOSUB Get_lblDS4
```

```
    ;Display current data label
    IF DBY(02EH)=0H THEN STRtoLCD1$=DataSet1$
    IF DBY(02EH)=28H THEN STRtoLCD1$=DataSet2$
    IF DBY(02EH)=50H THEN STRtoLCD1$=DataSet3$
    IF DBY(02EH)=78H THEN STRtoLCD1$=DataSet4$

    ;Send Mode Label to LCD line 1
    GOSUB String_To_LCD1

    ;Send Next/Previous Label to LCD line 2
    STRtoLCD2$="<Previous  Next>"
    GOSUB String_To_LCD2

    ;Enable INT0 to detect Matrix Traffic
    GOSUB Enable_INT0

    ;Enable INT1 to detect Keyboard traffic
    GOSUB Enable_INT1


;////////////////////////////////////////////////////////////////////////////////
;**Run****Run****Run****Run****Run****Run****Run****Run****Run****Run****Run****
;////////////////////////////////////////////////////////////////////////////////
;                            Run Mode Loop

Run_Loop:

#ASM
    ;Collision Reset
    ;Due to overlapping interrupts when multiple Keys on both the Keyboard and
    ;Matrix are depressed very quickly, there may be a timing issue where a
    ;bit is dropped. The collision reset prevents this from interfering with
    ;the normal operation by checking the status of Key_Bit_Num and
    ;Matrix_Bit_Num at the beginning of each loop

    ;Check Keyboard collisions
    MOV A,Key_Bit_Num                  ;get current Key bit number
    JZ RM_Key_OK                       ;see if it is zero
    CJNE A,Prev_Key_Num,RM_Key_OK      ;compare with value on last pass
    DEC Key_Checks_Left                ;if not zero and not different, count
                                       ;down one
    MOV A,Key_Checks_Left              ;test how many Key checks left
    JNZ RM_End_KColl
    ;If Key check counter reaches zero, reset Keyboard variables
    CLR Key_Sending
    MOV A,#00H
    MOV Key_Bit_Num,A
    MOV Key_Byte,A

RM_Key_OK:
    MOV Prev_Key_Num,A
    MOV A,#030H
    MOV Key_Checks_Left,A

RM_End_KColl:
        ;Check Matrix collisions
        MOV A,Matrix_Bit_Num                  ;get current Matrix bit number
        JZ RM_Matrix_OK                       ;see if it is zero
        CJNE A,Prev_Matrix_Num,RM_Matrix_OK ;compare with value on last pass
        DEC Matrix_Checks_Left                ;if not zero and not different, count
                                              ;down one
        MOV A,Matrix_Checks_Left              ;test how many Key checks left
        JNZ RM_End_MColl
        ;If Matrix check counter reaches zero, reset Keyboard variables
        CLR Matrix_Sending
        MOV A,#00H
        MOV Matrix_Bit_Num,A
        MOV Matrix_Byte,A

RM_Matrix_OK:
        MOV Prev_Matrix_Num,A
        MOV A,#030H
        MOV Matrix_Checks_Left,A

RM_End_MColl:

        ;Transfer bytes from lookup table using any bytes in the Matrix Buffer

RM_MB_Loop:
        ;Check to see if MB Current pointer = MB Last pointer
        ;If they are equal, there are no more bytes in Matrix Buffer to move
        MOV A,MB_Current                ;Put current Buffer position in ACC
        CJNE A,MB_Last,RM_MB_Move       ;If the two are not equal, go to the next
                                        ;step
        LJMP RM_KB_Loop                 ;Otherwise, skip to the Keyboard Buffer

RM_MB_Move:
        ;get the next byte in the Matrix Buffer
        INC MB_Current
        MOV A,MB_DPH
        MOV DPH,A
        MOV A,MB_Current
        MOV DPL,A
        MOVX A,@DPTR

        CJNE A,#0F0H,RM_MB_OF    ;Check to see if the byte = F0
        LJMP RM_MB_Release       ;if it does, jump to set the Release Flag

        ;*** If the received byte does not = F0 do the following
RM_MB_OF:
        ;make sure the byte is not for Buffer overflow (=FFH or 00H)
        CJNE A,#0FFH,RM_MB_OFCheck1
        LJMP RM_MB_Loop
RM_MB_OFCheck1:
        CJNE A,#0H,RM_MB_OFCheck2
        LJMP RM_MB_Loop
RM_MB_OFCheck2:
        ;Use the Matrix Byte to get the Pad number (0-79) from the
        ;lookup table
        MOV DPTR;#Pad_Lookup          ;Get the Code space address for the lookup
                                      ;table

        MOVC A,@A+DPTR                ;get the Matrix Pad number (0-79)

        ;Check for erroneous values
        ;The lookup table contains
        ;FFH for every byte value except those that are valid Matrix bytes
        CJNE A,#0FFH,RM_PadNum_OK
        LJMP RM_MB_Loop               ;otherwise there was an error, exit the loop

RM_PadNum_OK:

        ;Calculate the starting location for the Pad data
        ;first multiply the (Pad number in the accumulator) by 128 bytes/Pad
        MOV B,#080H ;128 bytes/Pad


        MUL AB      ;get Pad data location

        ;move the results to the data pointer
        MOV DPL,A

        MOV DPH,B

        ;Add the base address to the data pointer
        MOV A,#40H
        ADD A,DPH
        MOV DPH,A

        ;Add The Current Data Set offset to the address in DPTR
        MOV A,Data_Set
        ADD A,DPH
        MOV DPH,A

        ;get the first byte in the data table to detect the type of
        ;data stored there
        ;if first byte = 0 then no data
        ;if first byte = 1 then Single byte Key, one byte follows
        ;if first byte = 2 then Two byte Key, single byte follows
        ;(two Key bytes mean an E0 precedes the Key) To increase processing
        ;speed, the stored information only includes the Key byte, not the E0
        ;if first byte = 3 then Macro, the second byte holds a value between 0
        ;and 126 which indicates the number of sequential bytes in the Macro to
        ;transfer from the Pad storage location.

        MOVX A,@DPTR
        JZ RM_NoData        ;first byte = 0, not programmed
        DEC A
        JZ RM_SingleByte    ;first byte = 1, single byte Key
        DEC A
        JZ RM_DoubleByte    ;first byte = 2, two byte Key
        DEC A
        JZ RM_Macro         ;first byte = 3, Macro Key
        LJMP RM_MB_Loop     ;Error trap, program should not reach here
```

```
;*************************
RM_MB_Release:-
      SETB MB_Release
      LJMP RM_MB_Loop
;*************************
;*************************
RM_NoData:
      ;the Pad storage location is not programmed
      CLR MB_Release      ;Clear the release flag
      LJMP RM_MB_Loop     ;get the next byte
;*************************
;*************************
RM_SingleByte:
      ;This routine is used for Pads that are programmed as
      ;single byte Keys. SB Keys Send one byte (XX) when pressed
      ;and F0 XX when released.
      ;Get the next byte
      INC DPTR            ;point to the next byte
      MOVX A,@DPTR        ;transfer byte to the accumulator

      INC SB_Last         ;increment the pointer to add to the last byte in the
                          ;Send Buffer
      MOV DPL,SB_Last
      MOV DPH,SB_DPH

      ;if the Pad was pressed(Key down), jump over the F0 and
      ;just Send the Key byte
      JNB MB_Release, RM_SnglByteDown

      ;otherwise, the Key was released (Key up), so Send an F0 first
RM_SnglByteUp:
      ;Send an F0 byte
      MOV B,A             ;Store the Key byte
      MOV A,#0F0H         ;load accum with F0
      MOVX @DPTR,A        ;place F0 in the Send Buffer
      INC SB_Last         ;Increment the SB data pointer to prepare for the
      MOV DPL,SB_Last     ;Key byte

      MOV A,B             ;Restore the Key byte


RM_SnglByteDown:
      ;Send the Key byte
      MOVX @DPTR,A        ;place the Key byte in the Send Buffer

      CLR MB_Release      ;Clear the release flag
      LJMP RM_MB_Loop     ;get the next byte
;*************************
```

```
;*************************
RM_DoubleByte:
     ;This routine is used for Pads that are programmed as
     ;double byte Keys. DB Keys Send two bytes (E0 XX) when pressed
     ;and (E0 F0 XX) when released.

     ;Get the next byte
     INC DPTR             ;point to the next byte
     MOVX A,@DPTR         ;transfer byte to the accumulator

     INC SB_Last          ;increment the pointer to add to the last byte in the
                          ;Send Buffer
     NOP                  ;Check for a Buffer overflow (add later)
     MOV DPL,SB_Last
     MOV DPH,SB_DPH

     ;if the Pad was pressed(Key down), jump to the Key down routine
     JNB MB_Release, RM_DblByteDown

     ;otherwise, the Key was released (Key up), so Send an E0 F0 XX
RM_DblByteUp:

     MOV B,A              ;Store the Key byte

     ;Send E0 byte
     MOV A,#0E0H          ;load accum with E0
     MOVX @DPTR,A         ;place E0 in the Send Buffer
     INC SB_Last          ;Increment the SB data pointer to prepare for the
     MOV DPL,SB_Last      ;F0 byte

     ;Send F0 byte
     MOV A,#0F0H          ;load accum with F0
     MOVX @DPTR,A         ;place F0 in the Send Buffer
     INC SB_Last          ;Increment the SB data pointer to prepare for the
     MOV DPL,SB_Last      ;Key byte

     ;Send Byte
     MOV A,B              ;Restore the Key byte
     MOVX @DPTR,A         ;place the Key byte in the Send Buffer

     CLR MB_Release       ;Clear the release flag
     LJMP RM_MB_Loop      ;get the next byte

RM_DblByteDown:

     MOV B,A              ;Store the Key byte

     ;Send E0
     MOV A,#0E0H          ;load accum with E0
     MOVX @DPTR,A         ;place E0 in the Send Buffer
     INC SB_Last          ;Increment the SB data pointer to prepare for the
     MOV DPL,SB_Last      ;Key byte
```

```
        ;Send Byte
        MOV A,B             ;Restore the Key byte
        MOVX @DPTR,A        ;place the Key byte in the Send Buffer

        CLR MB_Release      ;Clear the release flag
        LJMP RM_MB_Loop     ;get the next byte
;*************************
;*************************
RM_Macro:
        ;This routine is used for Pads that are programmed as
        ;Macro Keys. Macro Keys send 0 to 126 sequential bytes
        ;when pressed, but do not send anything when released.
        ;the number of bytes to send is stored in the second
        ;byte of the Pad storage address, followed by the
        ;data bytes to Send.

        ;Check for Key release
        JNB MB_Release, RM_MacroDown

RM_MacroUp:
        CLR MB_Release      ;Clear the release flag
        LJMP RM_MB_Loop     ;get the next byte

RM_MacroDown:
        ;Get the number of bytes to Send in the Macro
        INC DPTR            ;point to the next byte
        MOVX A,@DPTR        ;transfer byte to the accumulator

        ;if zero bytes, jump to beginning of Matrix Buffer
        JNZ RM_StoreNumBytes
        LJMP RM_MB_Loop

RM_StoreNumBytes:
        ;store number of bytes to transfer in R0
        MOV R0,A

        ;*** Transfer the data to the Send Buffer (SB)
RM_MB_Transfer:

        INC SB_Last         ;increment the pointer to the last byte in the Buffer

        INC DPTR            ;Get the Next Byte to transfer to the SB
        MOVX A,@DPTR

        MOV R1,DPH          ;Temporarily store the DPTR to
        MOV R2,DPL          ;the bytes to transfer in R1 and R2

        MOV DPH,SB_DPH      ;Get the DPTR to the circular Send Buffer
        MOV DPL,SB_Last     ;in external memory

        MOVX @DPTR,A        ;Put the byte in the Send Buffer

        MOV DPH,R1          ;Restore the DPTR
        MOV DPL,R2          ;to the Send bytes


        DEC R0
        CJNE R0,#0H,RM_MB_Transfer
        LJMP RM_MB_Loop

;*************************


        ;Transfer bytes from Key Buffer to Send Buffer
RM_KB_Loop:

        ;Check to see if KB Current byte = KB Last byte
        ;If they are equal, no more bytes to Key Buffer to Send Buffer
        MOV A,KB_Current               ;Put current Buffer position in ACC
        CJNE A,KB_Last,RM_KB_Move      ;If the two are not equal, go to the next
;step
        LJMP RM_SB_Loop                ;Otherwise, skip to the Send Buffer

RM_KB_Move:
        ;transfer the next byte to the Send Buffer
        INC KB_Current            ;increment the pointer to the current byte in the
;Key Buffer
        INC SB_Last               ;Increment the pointer to the last byte in the
;Send Buffer

        ;get the byte
        MOV B,KB_DPH
        MOV DPH,B
        MOV B,KB_Current
        MOV DPL,B
        MOVX A,@DPTR

        ;store the byte
        MOV B,SB_DPH
        MOV DPH,B
        MOV B,SB_Last
        MOV DPL,B
        MOVX @DPTR,A

        ;Jump to beginning
        LJMP RM_KB_Loop


RM_SB_Loop:
        ;Transfer bytes from the Send Buffer to the CPU
        LCALL SB_Loop

#ASM_END
```

```
;///////////////////////////////////////////////////////////////////////////////
;**Run****Run****Run****Run****Run****Run****Run****Run****Run****Run****Run****
;///////////////////////////////////////////////////////////////////////////////
;                          Run Mode Check Mode


      Prog_Mode=XBY(0FFC0H)    ;Program control bits are memory mapped at
                               ;FFCXH

      ;Check for Button press
      chaTemp=Prog_Mode AND 1001_1100B
      IF (chaTemp>0) THEN Button_Delay=Button_Delay-1 ELSE Button_Delay=100

      IF Button_Delay>0 THEN GOTO RM_Prog_Mode

      ;Get the button that is pressed

      ;Check the Clear Button
      chaTemp=Prog_Mode AND 1000_0000B
      IF chaTemp=1000_0000B THEN GOTO Clear_Set

      ;Check the Previous Button
      chaTemp=Prog_Mode AND 0000_0100B
      IF chaTemp=0000_0100B THEN GOTO Previous_Set

      ;Check the Next Button
      chaTemp=Prog_Mode AND 0000_1000B
      IF chaTemp=0000_1000B THEN GOTO Next_Set

      ;Check the Label Button
      chaTemp=Prog_Mode AND 0001_0000B
      IF chaTemp=0001_0000B THEN GOTO Label_Set

RM_Prog_Mode:

      ;Check for system reset
      IF (PORT1 AND 0000_0100B)=0 THEN GOTO System_Reset

      ;Get the program mode
      Prog_Mode=Prog_Mode AND 0000_0011B

      IF Prog_Mode=0 THEN GOTO Macro_Mode
      IF Prog_Mode=2 THEN GOTO Key_Mode

GOTO Run_Loop
```

```
;//////////////////////////////////////////////////////////////////////////////////
;**Label****Label****Label****Label****Label****Label****Label****Label****Label
;//////////////////////////////////////////////////////////////////////////////////

Label_Set:
;This routine is used to change the label for the current data set
;from the Keyboard.  The labels are capital letters only
;(the shift and caps lock Keys are ignored), and
;no data is transmitted to the CPU.  After each
;letter is pressed on the Keyboard, it is added to the LCD.  Up to
;14 characters can be displayed on the LCD.
;The backspace Key deletes the previous letter and the
;enter Key accepts the new label and returns to the run mode.

        ;Disable the Matrix
        IE=IE AND 1111_1110B
        GOSUB Disable_INTO

        ;Initialize the Matrix Buffer
#ASM
        MOV A,#00H
        MOV MB_Current,A
        MOV MB_Last,A
#ASM_END

        ;Initialize the Key Buffer
#ASM
        MOV A,#00H
        MOV KB_Current,A
        MOV KB_Last,A
#ASM_END

        ;Initialize the Send Buffer
#ASM
        MOV A,#00H
        MOV SB_Current,A
        MOV SB_Last,A
        MOV SB_Prev_Byte,A

        CLR CAPS_Down
        CLR NUM_Down
        CLR SCROLL_Down
#ASM_END

        ;Initialize the Collision Check Variables
#ASM
        MOV A,#00H
        MOV Prev_Key_Num,A
        MOV Prev_Matrix_Num,A

        MOV A,#030H
        MOV Key_Checks_Left,A
        MOV Matrix_Checks_Left,A
#ASM_END
```

```
        ;reset and store the current label
        IF DBY(02EH)=0H THEN GOSUB Store_lblDS1
        IF DBY(02EH)=28H THEN GOSUB Store_lblDS2
        IF DBY(02EH)=50H THEN GOSUB Store_lblDS3
        IF DBY(02EH)=78H THEN GOSUB Store_lblDS4

LBL_Loop:
        ;Print the current label
        IF DBY(02EH)=0H THEN STRtoLCD1$=DataSet1$
        IF DBY(02EH)=28H THEN STRtoLCD1$=DataSet2$
        IF DBY(02EH)=50H THEN STRtoLCD1$=DataSet3$
        IF DBY(02EH)=78H THEN STRtoLCD1$=DataSet4$

        ;Send Mode Label to LCD line 1
        GOSUB String_To_LCD1

        ;Send Description to LCD line 2
        STRtoLCD2$="Type a new label":GOSUB String_To_LCD2


#ASM
LBL_GetNext:
        ;If Current and Last pointer are not equal,
        ;get the next byte in the Key Buffer

        ;Get bytes from the Keyboard input
        MOV A,KB_Current
        CJNE A,KB_Last,LBL_CheckByte
        AJMP LBL_GetNext

LBL_CheckByte:
        ;increment the pointer in the Keyboard Buffer
        INC KB_Current

        ;Get the next byte from the Keyboard Buffer
        MOV DPL,KB_Current
        MOV DPH,KB_DPH
        MOVX A,@DPTR

        ;check for E0
        CJNE A,#0E0H,LBL_No_E0
        SETB Flag_E0
        LJMP LBL_GetNext
LBL_No_E0:

        ;check for F0
        CJNE A,#0F0H,LBL_No_F0
        SETB Flag_F0
        LJMP LBL_GetNext
LBL_No_F0:
       ;check for backspace
       CJNE A,#066H,LBL_No_BS
              ;Check for E0 or F0
              JB Flag_E0, LBL_BSE0F0
              JB Flag_F0, LBL_BSE0F0

              ;Make sure the string pointer <>2
              ;(this ensures the user does not try to
              ;"backspace" past the colon in the label line)
              MOV A,LBL_PNTR
              CJNE A,#02H,LBL_GT3
                     CLR Flag_E0 ;Clear the E0 flag
                     CLR Flag_F0 ;Clear the F0 flag
                     LJMP LBL_GetNext  ;if pointer = 2, ignore and get next byte

              ;decrement string pointer by 1
LBL_GT3:
              DEC A               ;decrement pointer value
              MOV LBL_PNTR,A      ;store the new pointer value

              ;calculate character address
              MOV DPH,#0E7H       ;set the high byte to the label string storage
              ADD A,LBL_Offset    ;calculate the low byte to the label string
storage
              MOV DPL,A

              MOV A,#020H         ;store a blank space in accumulator
              MOVX @DPTR,A        ;put the blank space in the string
              LJMP LBL_End        ;jump to end of loop

LBL_BSE0F0:
              CLR Flag_E0 ;Clear the E0 flag
              CLR Flag_F0 ;Clear the F0 flag
              LJMP LBL_GetNext  ;Jump to beginning of loop

LBL_No_BS:

       ;check for enter
       CJNE A,#05AH,LBL_No_Enter

              ;return to run mode
              LJMP RM_Start

LBL_No_Enter:

       ;Add the new letter

       ;Check for E0 or F0
       JB Flag_E0, LBL_BSE0F0
       JB Flag_F0, LBL_BSE0F0
```

```
        ;get the display character for the Key
        MOV DPTR,#Label_Lookup          ;Get the Code space address for the lookup
                                        ;table
        MOVC A,@A+DPTR                  ;get the label

        ;check to make sure character is supported (only basic letter, number, and
        ;punctuation Keys)
        ;if there is no graphic associated with the Key
        ;the code 0FFH will be returned by the Label_Lookup table

        ;check for FF
        CJNE A,#0FFH,LBL_No_FF
              CLR Flag_E0
              CLR Flag_F0
              LJMP LBL_GetNext  ;otherwise, ignore non-label Key presses

LBL_No_FF:

        MOV B,A                         ;temporarily store the label value

        ;check to make sure the label pointer <>16
        MOV A,LBL_PNTR
        CJNE A,#016,LBL_LT17
              CLR Flag_E0
              CLR Flag_F0
              LJMP LBL_GetNext  ;if pointer = 16, ignore and get next byte

LBL_LT17:

        ;add the character to the display string
        MOV A,LBL_PNTR            ;get the current pointer
        ADD A,LBL_Offset          ;add the appropriate offset
        MOV DPL,A                 ;load the DPTR with the low byte
        MOV DPH,#0E7H             ;load the DPTR with the high byte

        ;store the new letter
        MOV A,B                   ;restore the letter value
        MOVX @DPTR,A              ;store the value

        ;increment the label pointer
        INC LBL_PNTR

        ;reset E0 and F0
        CLR Flag_F0
        CLR Flag_E0


LBL_End:

#ASM_END

        ;retrieve the current label
        IF DBY(02EH)=0H THEN GOSUB Get_lblDS1
        IF DBY(02EH)=28H THEN GOSUB Get_lblDS2
        IF DBY(02EH)=50H THEN GOSUB Get_lblDS3
        IF DBY(02EH)=78H THEN GOSUB Get_lblDS4
        ;Jump to the beginning of the loop
        GOTO LBL_Loop

;dsldsldsldsldsldsldsldsldsldsldsl
;***Store the label for Data Set 1
Store_lblDS1:
        ;reset the label
        DataSet1$="1:              "

        ;initialize the label pointer
        DBY(033H)=2

        ;initialize the label offset
        DBY(032H)=0

        ;store the string
        j=0
        FOR i=0E700H TO 0E70FH
                XBY(i)=ASC(DataSet1$,j)
                j=j+1
        NEXT i
Return ;{Store_lblDS1}

;***Empty the label for Data Set 1
Empty_lblDS1:
        ;reset the label
        DataSet1$="1:--Empty--     "

        ;store the string
        j=0
        FOR i=0E700H TO 0E70FH
                XBY(i)=ASC(DataSet1$,j)
                j=j+1
        NEXT i
Return ;{Empty_lblDS1}

;***Replace "Empty" with "No Label"
NoLBL_lblDS1:
;reset the label
        DataSet1$="1:--No Label--  "

        ;store the string
        j=0
        FOR i=0E700H TO 0E70FH
                XBY(i)=ASC(DataSet1$,j)
                j=j+1
        NEXT i
Return ;{NoLBL_lblDS1}
```

```
;***Get the label for Data Set 1
Get_lblDS1:
      ;Get the-string
      DataSet1$=""
      j=0
      FOR i=0E700H TO 0E70FH
            ASC(DataSet1$,j)=XBY(i)
            j=j+1
      NEXT i
Return ;{Get_lblDS1}
;ds1ds1ds1ds1ds1ds1ds1ds1ds1ds1ds1ds1

;ds2ds2ds2ds2ds2ds2ds2ds2ds2ds2ds2ds2
;***Store the label for Data Set 2
Store_lblDS2:
      ;reset the label
      DataSet2$="2:                "

      ;initialize the label pointer
      DBY(033H)=2

      ;initialize the label offset
      DBY(032H)=010H

      ;store the string
      j=0
      FOR i=0E710H TO 0E71FH
            XBY(i)=ASC(DataSet2$,j)
            j=j+1
      NEXT i
Return ;{Store_lblDS2}

;***Empty the label for Data Set 2
Empty_lblDS2:
      ;reset the label
      DataSet2$="2:--Empty--      "

      ;store the string
      j=0
      FOR i=0E710H TO 0E71FH
            XBY(i)=ASC(DataSet2$,j)
            j=j+1
      NEXT i
Return ;{Empty_lblDS2}

;***Replace "Empty" with "No Label"
NoLBL_lblDS2:
;reset the label
      DataSet2$="2:--No Label--  "

      ;store the string
      j=0
      FOR i=0E710H TO 0E71FH
            XBY(i)=ASC(DataSet2$,j)
            j=j+1
      NEXT i
Return ;{NoLBL_lblDS2}
```

```
;***Get the label for Data Set 2
Get_lblDS2:
        ;Get the string
        DataSet2$=""
        j=0
        FOR i=0E710H TO 0E71FH
               ASC(DataSet2$,j)=XBY(i)
               j=j+1
        NEXT i
Return ;{Get_lblDS2}
;ds2ds2ds2ds2ds2ds2ds2ds2ds2ds2ds2ds2

;ds3ds3ds3ds3ds3ds3ds3ds3ds3ds3ds3ds3
;***Store the label for Data Set 3
Store_lblDS3:
        ;reset the label
        DataSet3$="3:              "

        ;initialize the label pointer
        DBY(033H)=2

        ;initialize the label offset
        DBY(032H)=020H

        ;store the string
        j=0
        FOR i=0E720H TO 0E72FH
               XBY(i)=ASC(DataSet3$,j)
               j=j+1
        NEXT i
Return ;{Store_lblDS3}

;***Empty the label for Data Set 3
Empty_lblDS3:
        ;reset the label
        DataSet3$="3:--Empty--      "

        ;store the string
        j=0
        FOR i=0E720H TO 0E72FH
               XBY(i)=ASC(DataSet3$,j)
               j=j+1
        NEXT i
Return ;{Empty_lblDS3}

;***Replace "Empty" with "No Label"
NoLBL_lblDS3:
;reset the label
        DataSet3$="3:--No Label--   "

        ;store the string
        j=0
        FOR i=0E720H TO 0E72FH
               XBY(i)=ASC(DataSet3$,j)
               j=j+1
        NEXT i
Return ;{NoLBL_lblDS3}
```

```
;***Get the label for Data Set 3
Get_lblDS3:
      ;Get the-string
      DataSet3$=""
      j=0
      FOR i=0E720H TO 0E72FH
            ASC(DataSet3$,j)=XBY(i)
            j=j+1
      NEXT i
Return ;{Get_lblDS3}
;ds3ds3ds3ds3ds3ds3ds3ds3ds3ds3ds3ds3

;ds4ds4ds4ds4ds4ds4ds4ds4ds4ds4ds4ds4
;***Store the label for Data Set 4
Store_lblDS4:
      ;reset the label
      DataSet4$="4:                 "

      ;initialize the label pointer
      DBY(033H)=2

      ;initialize the label offset
      DBY(032H)=030H

      ;store the string
      j=0
      FOR i=0E730H TO 0E73FH
            XBY(i)=ASC(DataSet4$,j)
            j=j+1
      NEXT i
Return ;{Store_lblDS4}

;***Empty the label for Data Set 4
Empty_lblDS4:
      ;reset the label
      DataSet4$="4:--Empty--      "

      ;store the string
      j=0
      FOR i=0E730H TO 0E73FH
            XBY(i)=ASC(DataSet4$,j)
            j=j+1
      NEXT i
Return ;{Empty_lblDS4}

;***Replace "Empty" with "No Label"
NoLBL_lblDS4:
;reset the label
      DataSet4$="4:--No Label--  "

      ;store the string
      j=0
      FOR i=0E730H TO 0E73FH
            XBY(i)=ASC(DataSet4$,j)
            j=j+1
      NEXT i
Return ;{NoLBL_lblDS4}
```

```
;***Get the label for Data Set 4
Get_lblDS4:
      ;Get the-string
      DataSet4$=""
      j=0
      FOR i=0E730H TO 0E73FH
            ASC(DataSet4$,j)=XBY(i)
            j=j+1
      NEXT i
Return ;{Get_lblDS4}
;ds4ds4ds4ds4ds4ds4ds4ds4ds4ds4ds4ds4

;//////////////////////////////////////////////////////////////////////////////
;**Label****Label****Label****Label****Label****Label****Label****Label****Label
;//////////////////////////////////////////////////////////////////////////////

;//////////////////////////////////////////////////////////////////////////////
;**Clear****Clear****Clear****Clear****Clear****Clear****Clear****Clear****Clear
;//////////////////////////////////////////////////////////////////////////////
;This routine wipes all programmed Matrix Pads in the current Data set.
;This also resets the current Label to "--Empty--"

Clear_Set:

      ;Disable the Keyboard and Matrix
      GOSUB Disable_INT0
      GOSUB Disable_INT1

      ;Initialize the Matrix Buffer
#ASM
      MOV A,#00H
      MOV MB_Current,A
      MOV MB_Last,A
#ASM_END

      ;Initialize the Key Buffer
#ASM
      MOV A,#00H
      MOV KB_Current,A
      MOV KB_Last,A
#ASM_END

      ;Initialize the Send Buffer.
#ASM
      MOV A,#00H
      MOV SB_Current,A
      MOV SB_Last,A
      MOV SB_Prev_Byte,A

      CLR CAPS_Down
      CLR NUM_Down
      CLR SCROLL_Down
#ASM_END


      ;Initialize the Collision Check Variables
#ASM
```

```
        MOV A,#00H
        MOV Prev_Key_Num,A
        MOV Prev_Matrix_Num,A

        MOV A,#030H
        MOV Key_Checks_Left,A
        MOV Matrix_Checks_Left,A
#ASM_END

        ;Query the user to verify he wants to erase the current data set
        IF DBY(02EH)=00H THEN STRtoLCD1$="Erase Set #1 ?    "
        IF DBY(02EH)=28H THEN STRtoLCD1$="Erase Set #2 ?    "
        IF DBY(02EH)=50H THEN STRtoLCD1$="Erase Set #3 ?    "
        IF DBY(02EH)=78H THEN STRtoLCD1$="Erase Set #4 ?    "

        ;Send Mode Label to LCD line 1
        GOSUB String_To_LCD1

        ;Send Description to LCD line 2
        STRtoLCD2$="<Yes          No>" : GOSUB String_To_LCD2

Clear_YorN:
        ;Wait for Yes (Left Button) or No (Right Button)
        chaTemp=XBY(0FFC0H)      ;buttons are memory mapped at FFCXH

        ;Check for Button press
        chaTemp=chaTemp AND 0000_1100B

        IF chaTemp=0000_0100B THEN GOTO Clear_Yes
        IF chaTemp=0000_1000B THEN GOTO Clear_No

        GOTO Clear_YorN

Clear_Yes:
        ;Clear Display
        ;Erase LCD line 1
        STRtoLCD1$="                " : GOSUB String_To_LCD1

        ;Erase LCD line 2
        STRtoLCD2$="                " : GOSUB String_To_LCD2

        ;Erase memory block for current data set
        IF DBY(02EH)=0H THEN l=04000H
        IF DBY(02EH)=0H THEN m=067FFH

        IF DBY(02EH)=28H THEN l=06800H
        IF DBY(02EH)=28H THEN m=089FFH

        IF DBY(02EH)=50H THEN l=09000H
        IF DBY(02EH)=50H THEN m=0B7FFH

        IF DBY(02EH)=78H THEN l=0B800H
        IF DBY(02EH)=78H THEN m=0DFFFH
        FOR i=1 to m
             XBY(i)=00H
        NEXT i

        ;Set to "--Empty--"
        ;and store the current label
        IF DBY(02EH)=0H THEN GOSUB Empty_lblDS1
        IF DBY(02EH)=28H THEN GOSUB Empty_lblDS2
        IF DBY(02EH)=50H THEN GOSUB Empty_lblDS3
        IF DBY(02EH)=78H THEN GOSUB Empty_lblDS4

 Clear_YesLoop:
        ;Wait for release of Yes Button
        chaTemp=XBY(0FFC0H)     ;buttons are memory mapped at FFCXH

        ;Check for Button press
        chaTemp=chaTemp AND 0000_1100B

        IF chaTemp<>0 THEN GOTO Clear_YesLoop

        ;Goto Start of Run Loop
        GOTO Run_Mode

 Clear_No:
        ;Wait for release of No Button
        chaTemp=XBY(0FFC0H)     ;buttons are memory mapped at FFCXH

        ;Check for Button press
        chaTemp=chaTemp AND 0000_1100B

        IF chaTemp<>0 THEN GOTO Clear_No

        ;Goto Start of Run Loop
        GOTO Run_Mode


;////////////////////////////////////////////////////////////////////////////////
;**Clear****Clear****Clear****Clear****Clear****Clear****Clear****Clear****Clear
;////////////////////////////////////////////////////////////////////////////////

;////////////////////////////////////////////////////////////////////////////////
;**Next****Next****Next****Next****Next****Next****Next****Next****Next****Next*
;////////////////////////////////////////////////////////////////////////////////

;This routine activates the next data set (1->2, 2->3, 3->4, 4->1)
;Called by pressing the right button on the device.
Next_Set:
       ;Clear the current label
       STRtoLCD1$="                    " : GOSUB String_to_LCD1

       ;Load the next data set
       IF DBY(02EH)=0H THEN DBY(02EH)=028H : GOTO NEXT_Wait
       IF DBY(02EH)=028H THEN DBY(02EH)=050H : GOTO NEXT_Wait
       IF DBY(02EH)=050H THEN DBY(02EH)=078H : GOTO NEXT_Wait
       IF DBY(02EH)=078H THEN DBY(02EH)=0H : GOTO NEXT_Wait

NEXT_Wait:
      ;Wait for Key Release
      Button_Delay=100

NEXT_Loop:
      Prog_Mode=XBY(0FFC0H)
      chaTemp=Prog_Mode AND 1001_1100B
      IF (chaTemp=0) THEN Button_Delay=Button_Delay-1 ELSE Button_Delay=100
6000
      IF (Button_Delay=0) THEN GOTO Run_Mode

      GOTO NEXT_Loop

;////////////////////////////////////////////////////////////////////////////////////
;**Next****Next****Next****Next****Next****Next****Next****Next****Next****Next*
;////////////////////////////////////////////////////////////////////////////////////

;////////////////////////////////////////////////////////////////////////////////////
;**Previous****Previous****Previous****Previous****Previous****Previous****
;////////////////////////////////////////////////////////////////////////////////////
;This routine activates the previous data set (1->4, 2->1, 3->2, 4->3)
;Called by pressing the left button on the device.
Previous_Set:
      ;Clear the current label
      STRtoLCD1$="                    " : GOSUB String_to_LCD1

      ;Load the next data set
      IF DBY(02EH)=0H THEN DBY(02EH)=078H : GOTO PREV_Wait
      IF DBY(02EH)=028H THEN DBY(02EH)=0H : GOTO PREV_Wait
      IF DBY(02EH)=050H THEN DBY(02EH)=028H : GOTO PREV_Wait
      IF DBY(02EH)=078H THEN DBY(02EH)=050H : GOTO PREV_Wait

PREV_Wait:

      ;Wait for Key Release
      Button_Delay=100

PREV_Loop:
      Prog_Mode=XBY(0FFC0H)
      chaTemp=Prog_Mode AND 1001_1100B
      IF (chaTemp=0) THEN Button_Delay=Button_Delay-1 ELSE Button_Delay=100
7000
      IF (Button_Delay=0) THEN GOTO Run_Mode
      GOTO PREV_Loop

;////////////////////////////////////////////////////////////////////////////////////
;**Previous****Previous****Previous****Previous****Previous****Previous****
;////////////////////////////////////////////////////////////////////////////////////
```

```
;///////////////////////////////////////////////////////////////////////////////
;**Key****Key****Key****Key****Key****Key****Key****Key****Key****Key****Key****
;///////////////////////////////////////////////////////////////////////////////
;                            Key Mode Startup

Key_Mode:

      ;Initialize the Matrix Buffer
#ASM
KM_Start:
      MOV A,#00H
      MOV MB_Current,A
      MOV MB_Last,A
      CLR MB_Release
#ASM_END

      ;Initialize the Key Buffer
#ASM
      MOV A,#00H
      MOV KB_Current,A
      MOV KB_Last,A
#ASM_END


      ;Initialize the Code Variables
#ASM
      CLR Flag_E0
      CLR Flag_F0
      CLR Flag_M_F0

      MOV A,#0FFH
      MOV Matrix_Code,A
      MOV Key_Code,A
#ASM_END

      B_Key_Code=0FFH
      B_E0_Flag=0
      B_Matrix_Code=0FFH


      ;Send Label to LCD line 1
      STRtoLCD1$="LAST KEY:       "
      GOSUB String_To_LCD1

      ;Send Label to LCD line 2
      STRtoLCD2$="ASSIGNED TO:    "
      GOSUB String_To_LCD2

      ;Enable INT0 to detect Matrix Traffic
      GOSUB Enable_INT0

      ;Enable INT1 to detect Keyboard traffic
      GOSUB Enable_INT1
```

```
;//////////////////////////////////////////////////////////////////////////////
;**Key****Key****Key****Key****Key****Key****Key****Key****Key****Key****Key****
;//////////////////////////////////////////////////////////////////////////////
;                         Key Mode Loop     :

Key_Loop:

#ASM

      ;Get the next byte in the Matrix Buffer
KM_MB_Loop:
      ;Check to see if MB Current pointer = MB Last pointer
      ;If they are equal, there are no more bytes in Matrix Buffer to get
      MOV A,MB_Current              ;Put current Buffer position in ACC
      CJNE A,MB_Last,KM_MB_Move     ;If the two are not equal, go to the next
                                    ;step
      LJMP KM_KB_Loop               ;Otherwise, skip to the Keyboard Buffer

KM_MB_Move:
      ;get the next byte in the Matrix Buffer
      INC MB_Current
      MOV A,MB_DPH
      MOV DPH,A
      MOV A,MB_Current
      MOV DPL,A
      MOVX A,@DPTR

      CJNE A,#0F0H,KM_MB_ChkFlag    ;Check to see if the byte = F0
      LJMP KM_MB_Release            ;if it does, jump to set the Release Flag

      ;*** If the received byte does not = F0 do the following
KM_MB_ChkFlag:
      ;Check to see if Release flag is set
      JNB MB_Release,KM_MB_Store    ;if release bit is not set, then store value
      CLR MB_Release                ;otherwise, clear the release bit and jump
      LJMP KM_MB_Loop               ;back to the beginning of the loop

KM_MB_Store:
      MOV Matrix_Code,A       ;store the Matrix code
      LJMP KM_MB_Loop         ;Jump back to the beginning of the loop


      ;Get the next byte in the Key Buffer
KM_KB_Loop:

      ;Check to see if KB Current byte = KB Last byte
      ;If they are equal, no more bytes are in the Key Buffer
      MOV A,KB_Current              ;Put current Buffer position in ACC
      CJNE A,KB_Last,KM_KB_Move     ;If the two are not equal, go to the next
                                    ;step
      LJMP KM_BASIC_Block           ;Otherwise, skip to the Keyboard Mode BASIC
                                    ;block
```

```
KM_KB_Move:
        ;Get the next byte
        INC KB_Current           ;increment the pointer to the Key Buffer
        MOV B,KB_DPH
        MOV DPH,B
        MOV B,KB_Current
        MOV DPL,B
        MOVX A,@DPTR

        ;Check for Overflow code
        ;make sure the byte is not for Buffer overflow (=FFH or 00H)
        CJNE A,#0FFH,KM_KB_OFCheck1
        LJMP KM_KB_Loop
KM_KB_OFCheck1:
        CJNE A,#0H,KM_KB_OFCheck2
        LJMP KM_KB_Loop
KM_KB_OFCheck2:

        ;Check for F0
        CJNE A,#0F0H,KM_KB_E0   ;Check to see if the byte = F0
        LJMP KM_KB_Release      ;if it does, jump to set the F0 Flag

KM_KB_E0:
        ;Check for E0
        CJNE A,#0E0H,KM_KB_ChkFlag     ;Check to see if the byte = E0
        LJMP KM_KB_Extend              ;if it does, jump to set the E0 Flag

KM_KB_ChkFlag:
        ;Check to see if F0 flag is set
        JNB Flag_F0,KM_KB_Store        ;if release bit is not set, then store value
        CLR Flag_F0                    ;otherwise, clear the F0 flag
        CLR Flag_E0                    ;and the E0 Flag
        LJMP KM_MB_Loop                ;and jump back to the beginning of the loop

KM_KB_Store:
        MOV Key_Code,A          ;store the Matrix code

KM_BASIC_Block:
#ASM_END

        ;Switch to BASIC for string manipulation

        ;Get and Display any new Keyboard Key press

        ;Check if Key_Code = FF
        IF DBY(02FH)=0FFH THEN GOTO KM_Check_Matrix

        ;if Key code DOES NOT equal FF, then a new Keyboard Key press was recorded

        ;Transfer value from assembly to BASIC variable
        B_Key_Code=DBY(02FH)
        ;Get the new Keypress string
        ;Test for E0 Keys
        IF (DBY(022H) AND 0100_0000B) = 0100_0000B THEN GOSUB Get_E0_Keystring


        ;Test for non-E0 Keys
        IF (DBY(022H) AND 0100_0000B) = 0000_0000B THEN GOSUB Get_Keystring

        ;Display the Keypress value on line 1
        GOSUB String_To_LCD1

        ;Reset line 2 (the "Assigned To:") string
        STRtoLCD2$="ASSIGNED TO:     "
        GOSUB String_To_LCD2

#ASM
        ;Set Assembly Key_Code to FF
        MOV A,#0FFH
        MOV Key_Code,A

        ;Clear Assembly E0_Flag
        CLR Flag_E0
#ASM_END

KM_Check_Matrix:

        ;Check if Matrix_Code = FF
        IF DBY(02DH)=0FFH THEN GOTO KM_Check_Mode
        ;if Matrix code DOES NOT equal FF, then a new Matrix Key press was
        ;recorded

        ;Check if a Keyboard code is already loaded
        IF B_Key_Code=0FFH THEN GOTO KM_Check_Mode
        ;if Keyboard code DOES NOT equal FF, then a Keyboard Key is already loaded

        ;Transfer value from assembly to BASIC variable
        B_Matrix_Code=DBY(02DH)

        ;Get the string that describes the Matrix position
        GOSUB Get_Matrix_Pos

        ;Create the label for the Matrix position
        STRtoLCD2$="ASSIGNED TO:" + Matrix_Position$
        GOSUB String_To_LCD2

        ;Pass the current value of the Key code and
        ;E0 flag to the assembly routine

        ;Store the current BASIC Key code value in Pass_Key_Value
        DBY(040H)=B_Key_Code
        ;Store the current BASIC E0 Flag value in Pass_E0_Value
        DBY(041H)=B_E0_Flag
        ;If the current label is "--Empty--" then change it to
        ;"--No Label--" to indicate at least one Pad is programmed
        ;Check current data set for "empty"
        IF DBY(02EH)=0H THEN GOSUB Check_DS1_Empty
        IF DBY(02EH)=28H THEN GOSUB Check_DS2_Empty
        IF DBY(02EH)=50H THEN GOSUB Check_DS3_Empty
        IF DBY(02EH)=78H THEN GOSUB Check_DS4_Empty


        ;Switch to Assembly to store the code at the specified Pad location
#ASM

        MOV A,Matrix_Code        ;Get the Matrix code value
        MOV DPTR,#Pad_Lookup     ;Get the Code space address for the lookup table

        MOVC A,@A+DPTR           ;get the Matrix Pad number (0-79)

        ;Check for erroneous values
        ;The lookup table contains
        ;FFH for every byte value except those that are valid Matrix bytes
        CJNE A,#0FFH,KM_PadNum_OK
        LJMP KM_Store_End        ;otherwise exit on error

KM_PadNum_OK:

        ;Calculate the starting location for the Pad data
        ;first multiply the (Pad number in the accumulator) by 128 bytes/Pad
        MOV B,#080H ;128 bytes/Pad

        MUL AB       ;get Pad data location

        ;move the results to the data pointer
        MOV DPL,A
        MOV DPH,B

        ;Add the base address to the data pointer
        MOV A,#40H
        ADD A,DPH
        MOV DPH,A

        ;Add The Current Data Set offset to the address in DPTR
        MOV A,Data_Set
        ADD A,DPH
        MOV DPH,A

        ;(Before entering this section, the current E0 flag
        ;status was passed to register R0.)
        ;Move E0 Flag status to accumulator
        MOV A,Pass_E0_Value

        ;If E0 Flag for current byte is 0 then store a 1 in the first
        ;Pad location.
        CJNE A,#00H,KM_E0_1
KM_E0_0:
      MOV A,#01H          ;Move the value 1 to acc
      MOVX @DPTR,A        ;Store 1 in the first Pad location

      LJMP KM_Store_Byte       ;Jump to store the second byte (Key code value)

      ;if E0 Flag for current byte is 1, then store a 2 in the first
      ;Pad location.
KM_E0_1:

      MOV A,#02H          ;Move the value 2 to acc
      MOVX @DPTR,A        ;Store 2 in the first Pad location

KM_Store_Byte:

      ;The current Key code is stored in register R1
      INC DPTR                 ;increment the data pointer
      MOV A,Pass_Key_Value     ;move the cuurent Key code to acc
      MOVX @DPTR,A             ;store the Key code in the second Pad location

KM_Store_End:
#ASM_END

      ;Set Assembly Matrix Code to FF
      DBY(02DH)=0FFH

      GOTO KM_Check_Mode
;*************************
#ASM
KM_MB_Release:
      SETB MB_Release
      LJMP KM_MB_Loop
#ASM_END
;*************************

;*************************
#ASM
KM_KB_Release:
      SETB Flag_F0
      LJMP KM_KB_Loop
#ASM_END
;*************************

;*************************
#ASM
KM_KB_Extend:
      SETB Flag_E0
      LJMP KM_KB_Loop
#ASM_END
;*************************
```

```
;****************************************************************
Get_Keystring:
       ;Get the string that describes the Key pressed
       GOSUB Get_Keys

       ;Create the label for the Key
       STRtoLCD1$="LAST KEY:" + Key_Label$

       ;Clear the BASIC E0 Flag
       B_E0_Flag=0

RETURN ; {Get_Keystring}
;****************************************************************


;****************************************************************
Get_E0_Keystring:
       ;Get the string that describes the Key pressed
       GOSUB Get_E0_Keys

       ;Create the label for the Key
       STRtoLCD1$="LAST KEY:" + Key_Label$

       ;Set the BASIC E0 Flag
       B_E0_Flag=1

RETURN ; {Get_E0_Keystring}
;****************************************************************

;****************************************************************
       ;Full string equivalence checks (If A$=B$ THEN ...) are not
       ;supported by the compiler.  To test if the current
       ;data label = "1:--Empty--", the 6th element is compared to
       ;the value "m".  The user may only submit CAPITAL letters
       ;for labels, so the presence of a lowercase "m" at this
       ;location will identify the string as "1:--Empty--"
Check_DS1_Empty:
       strTemp$="m"
       IF ASC(strTemp$,0)=ASC(Dataset1$,5) THEN GOSUB NoLBL_lblDS1
RETURN ; {Check_DS1_Empty}
;****************************************************************

;****************************************************************
Check_DS2_Empty:
       strTemp$="m"
       IF ASC(strTemp$,0)=ASC(Dataset2$,5) THEN GOSUB NoLBL_lblDS2
RETURN ; {Check_DS2_Empty}
;****************************************************************

;****************************************************************
Check_DS3_Empty:
       strTemp$="m"
       IF ASC(strTemp$,0)=ASC(Dataset3$,5) THEN GOSUB NoLBL_lblDS3
RETURN ; {Check_DS3_Empty}
;****************************************************************
```

```
;***************************************************************
Check_DS4_Empty:
      strTemp$="m"
      IF ASC(strTemp$,0)=ASC(Dataset4$,5) THEN GOSUB NoLBL_lblDS4
RETURN ; {Check_DS4_Empty}
;***************************************************************


;////////////////////////////////////////////////////////////////////////////////
;**Key****Key****Key****Key****Key****Key****Key****Key****Key****Key****Key****
;////////////////////////////////////////////////////////////////////////////////
;                        Key Mode Check Mode

KM_Check_Mode:

      Prog_Mode=XBY(0FFC0H)         ;Program control bits are memory mapped at
                                    ;FFCXH

      Prog_Mode=Prog_Mode AND 0000_0011B

      IF Prog_Mode=0 THEN GOTO Macro_Mode
      IF Prog_Mode=1 THEN GOTO Run_Mode

      Goto Key_Loop


;////////////////////////////////////////////////////////////////////////////////
;**Macro****Macro****Macro****Macro****Macro****Macro****Macro****Macro****Macro
;////////////////////////////////////////////////////////////////////////////////
;                        Macro Mode Startup

Macro_Mode:

      ;Initialize the Matrix Buffer
#ASM
MM_Start:
      MOV A,#00H
      MOV MB_Current,A
      MOV MB_Last,A
      CLR MB_Release
#ASM_END

      ;Initialize the Key Buffer
#ASM
      MOV A,#00H
      MOV KB_Current,A
      MOV KB_Last,A
#ASM_END
```

```
      ;Initialize the Send Buffer

#ASM
      MOV A,#00H
      MOV SB_Current,A
      MOV SB_Last,A
      MOV SB_Prev_Byte,A

      CLR CAPS_Down
      CLR NUM_Down
      CLR SCROLL_Down
#ASM_END


      ;Initialize the Macro Buffer
#ASM
      MOV A,#00H
      MOV AB_Length,A
#ASM_END

      ;Initialize the Code Variables
#ASM
      MOV A,#0FFH
      MOV Matrix_Code,A
#ASM_END

      B_Matrix_Code=0FFH

      ;Initialize the Collision Check Variables
#ASM
      MOV A,#00H
      MOV Prev_Key_Num,A
      MOV Prev_Matrix_Num,A

      MOV A,#030H
      MOV Key_Checks_Left,A
      MOV Matrix_Checks_Left,A
#ASM_END

      ;Send Record Label to LCD line 1
      STRtoLCD1$="RECORD: ........"
      GOSUB String_To_LCD1

      ;Clear LCD line 2
      STRtoLCD2$="                "
      GOSUB String_To_LCD2

      ;Enable INT0 to detect Matrix Traffic
      GOSUB Enable_INT0

      ;Enable INT1 to detect Keyboard traffic
      GOSUB Enable_INT1
```

```
;///////////////////////////////////////////////////////////////////////////////
;**Macro****Macro****Macro****Macro****Macro****Macro****Macro****Macro****Macro
;///////////////////////////////////////////////////////////////////////////////
;                          Macro Mode Loop

Macro_Loop:

#ASM
      ;Collision Reset
      ;Due to overlapping interrupts when multiple Keys on both the Keyboard and
      ;Matrix are depressed very quickly, there may be a timing issue where a
      ;bit is dropped. The collision reset prevents this from interfering with
      ;the normal operation by checking the status of Key_Bit_Num and
      ;Matrix_Bit_Num at the beginning of each loop

      ;Check Keyboard collisions
      MOV A,Key_Bit_Num                  ;get current Key bit number
      JZ AM_Key_OK                       ;see if it is zero
      CJNE A,Prev_Key_Num,AM_Key_OK      ;compare with value on last pass
      DEC Key_Checks_Left                ;if not zero and not different, count
                                         ;down one
      MOV A,Key_Checks_Left              ;test how many Key checks left
      JNZ AM_End_KColl
      ;If Key check counter reaches zero, reset Keyboard variables
      CLR Key_Sending
      MOV A,#00H
      MOV Key_Bit_Num,A
      MOV Key_Byte,A

AM_Key_OK:
      MOV Prev_Key_Num,A
      MOV A,#030H
      MOV Key_Checks_Left,A

AM_End_KColl:


      ;Check Matrix collisions
      MOV A,Matrix_Bit_Num               ;get current Matrix bit number
      JZ AM_Matrix_OK                    ;see if it is zero
      CJNE A,Prev_Matrix_Num,AM_Matrix_OK ;compare with value on last pass
      DEC Matrix_Checks_Left             ;if not zero and not different, count
                                         ;down one
      MOV A,Matrix_Checks_Left.          ;test how many Key checks left
      JNZ AM_End_MColl
      ;If Matrix check counter reaches zero, reset Keyboard variables
      CLR Matrix_Sending
      MOV A,#00H
      MOV Matrix_Bit_Num,A
      MOV Matrix_Byte,A

AM_Matrix_OK:
      MOV Prev_Matrix_Num,A
      MOV A,#030H
      MOV Matrix_Checks_Left,A

AM_End_MColl:
      ;Transfer bytes from Key Buffer to Send Buffer
AM_KB_Loop:

      ;Check to see if KB Current byte = KB Last byte
      ;If they are equal, no more bytes to Key Buffer to Send Buffer
      MOV A,KB_Current            ;Put current Buffer position in ACC
      CJNE A,KB_Last,AM_KB_Full   ;If the two are not equal, go to the next
                                  ;step
      LJMP AM_MB_Loop             ;Otherwise, skip to the Send Buffer

AM_KB_Full:
      ;Check if Macro Buffer is already full
      MOV A,AB_Length
      CJNE A,#07EH,AM_KB_Move

      ;if Buffer is full, increment the current byte in the
      ;Key Buffer, but DO NOT place it in the Send Buffer
      ;When the Macro Buffer is full, no further
      ;Keystrokes on the Keyboard will be transfered to
      ;the CPU.  The user must either bind the Macro to a pad
      ;location, or exit the Macro recording mode to use the
      ;Keyboard again.

      INC KB_Current

      ;Jump to beginning of Macro Key Loop
      LJMP AM_KB_Loop

AM_KB_Move:
      ;get the next byte
      INC KB_Current     ;increment the pointer to the next byte in the Key
                         ;Buffer
      MOV B,KB_DPH
      MOV DPH,B
      MOV B,KB_Current
      MOV DPL,B
      MOVX A,@DPTR

      ;Transfer the next byte to the Macro Buffer
      INC AB_Length      ;Increment the Macro Length Counter
      MOV B,AB_DPH
      MOV DPH,B
      MOV B,AB_Length
      MOV DPL,B
      MOVX @DPTR,A

      ;transfer the next byte to the Send Buffer
      INC SB_Last        ;Increment the pointer to the last byte in the Send
                         ;Buffer
      MOV B,SB_DPH
      MOV DPH,B
      MOV B,SB_Last
      MOV DPL,B
      MOVX @DPTR,A
      ;Jump to beginning of the Macro Keyboard loop
      LJMP AM_KB_Loop

;Get the next byte in the Matrix Buffer
AM_MB_Loop:
      ;Check to see if MB Current pointer = MB Last pointer
      ;If they are equal, there are no more bytes in Matrix Buffer to get
      MOV A,MB_Current               ;Put current Buffer position in ACC
      CJNE A,MB_Last,AM_MB_Move      ;If the two are not equal, go to the next
                                     ;step
      LJMP AM_SB_Loop                ;Otherwise, skip to the Send Buffer

AM_MB_Move:
      ;get the next byte in the Matrix Buffer
      INC MB_Current
      MOV A,MB_DPH
      MOV DPH,A
      MOV A,MB_Current
      MOV DPL,A
      MOVX A,@DPTR

      CJNE A,#0F0H,AM_MB_ChkFlag     ;Check to see if the byte = F0
      LJMP AM_MB_Release             ;if it does, jump to set the Release Flag

      ;*** If the received byte does not = F0 do the following
AM_MB_ChkFlag:
      ;Check to see if Release flag is set
      JNB MB_Release,AM_MB_Store     ;if release bit is not set, then store value
      CLR MB_Release                 ;otherwise, clear the release bit and jump
      LJMP AM_MB_Loop                ;back to the beginning of the loop

AM_MB_Store:
      MOV Matrix_Code,A         ;store the Matrix code
      LJMP AM_MB_Loop           ;Jump back to the beginning of the loop

AM_SB_Loop:
      ;Use the SB_Loop subroutine to transfer the bytes
      ;in the Send Buffer to the CPU
      LCALL SB_Loop

#ASM_END

      ;Switch to BASIC for string manipulation

      ;generate the current Macro length strings
      GOSUB Disp_Mac_Len

      ;Display Bar line
      IF DBY(01EH)>0 THEN GOSUB String_To_LCD1

      ;Display Asterisk line
      IF DBY(01EH)>0 THEN GOSUB String_To_LCD2

      ;Check if Matrix_Code = FF
      IF DBY(02DH)=0FFH THEN GOTO AM_Check_Mode
      ;if Matrix code DOES NOT equal FF, then a new Matrix Key press was
      ;recorded
      ;Check if Macro has any data in it
      IF DBY(01EH)=0H THEN GOTO KM_Check_Mode
      ;if Macro length > 0 then bytes need to be transferred.

      ;Transfer Matrix Code value from assembly to BASIC variable
      B_Matrix_Code=DBY(02DH)

      ;Get the string that describes the Matrix position
      GOSUB Get_Matrix_Pos

      ;Create the label for the Matrix position
      STRtoLCD1$="RECORDED MACRO  "
      GOSUB String_To_LCD1

      STRtoLCD2$="ASSIGNED TO:" + Matrix_Position$
      GOSUB String_To_LCD2

      ;If the current label is "--Empty--" then change it to
      ;"--No Label--" to indicate at least one Pad is programmed
      ;Check current data set for "empty"
      IF DBY(02EH)=0H THEN GOSUB Check_DS1_Empty
      IF DBY(02EH)=28H THEN GOSUB Check_DS2_Empty
      IF DBY(02EH)=50H THEN GOSUB Check_DS3_Empty
      IF DBY(02EH)=78H THEN GOSUB Check_DS4_Empty

      ;Switch to Assembly to store the Macro at the specified pad location
#ASM

      MOV A,Matrix_Code          ;Get the Matrix code value
      MOV DPTR,#Pad_Lookup     ;Get the Code space address for the lookup table

      MOVC A,@A+DPTR             ;get the Matrix Pad number (0-79)

      ;Check for erroneous values
      ;The lookup table contains
      ;FFH for every byte value except those that are valid Matrix bytes
      CJNE A,#0FFH,AM_PadNum_OK
      LJMP AM_Store_End           ;otherwise exit on error

AM_PadNum_OK:

      ;Calculate the starting location for the pad data
      ;first multiply the (Pad number in the accumulator) by 128 bytes/Pad

      MOV B,#080H ;128 bytes/Pad
      MUL AB      ;get Pad data location

      ;move the results to the datapointer
      MOV DPL,A
      MOV DPH,B

      ;Add the base address to the data pointer
      MOV A,#40H
      ADD A,DPH
      MOV DPH,A
```

```
        ;Add The Current Data Set offset to the address in DPTR
        MOV A,Data_Set
        ADD A,DPH
        MOV DPH,A

        ;store a 3 in location 1 to indicate a Macro
        MOV A,#03H          ;Move the value 3 to acc
        MOVX @DPTR,A        ;Store 3 in the first Pad location to
                            ;indicate a Macro

        ;store the number of bytes in the Macro in location 2
        INC DPTR            ;Point to location 2
        MOV A,AB_Length     ;Load the current length of Macro Buffer
        MOVX @DPTR,A        ;store the Macro length in pos 2

        ;store the Macro in location 3 through 128, as needed

        ;Point to the zero byte in the Macro source
        ;Note, there is no byte located at 0E800H
        ;the first recorded byte is stored at 0E801H
        ;to correspond to the Macro length variable
        ;(i.e. start counting at 1, not at 0)
        MOV R1,#0H

AM_Macro_Xfer:
        ;Increment the Macro counter
        INC R1

        ;store the data pointer to the destination
        ;in r2 and r3
        MOV R2,DPL
        MOV R3,DPH

        ;Load the data pointer with the Macro Buffer address
        MOV DPL,R1
        MOV DPH,AB_DPH

        ;Get the next Macro Buffer value
        MOVX A,@DPTR

        ;Reload the destination pointer
        MOV DPL,R2
        MOV DPH,R3

        ;increment the destination pointer
        INC DPTR

        ;Store the value at the destination
        MOVX @DPTR,A

        ;Check the value of the Macro counter
        ;to see if it equals the Macro
        ;length
        MOV A,R1
        CJNE A,AB_Length,AM_Macro_Xfer
```

```
AM_Store_End:

      ;Set Assembly Matrix Code to FF
      MOV A,#0FFH
      MOV Matrix_Code,A

      ;Set Macro length to zero
      MOV A,#0H
      MOV AB_Length,A

#ASM_END

      GOTO AM_Check_Mode

;*************************
#ASM
AM_MB_Release:
      SETB MB_Release
      LJMP AM_MB_Loop
#ASM_END
;*************************

;//////////////////////////////////////////////////////////////////////////////
;**Macro****Macro****Macro****Macro****Macro****Macro****Macro****Macro****Macro
;//////////////////////////////////////////////////////////////////////////////
;                        Macro Mode Check Mode

AM_Check_Mode:

Prog_Mode=XBY(0FFC0H)                ;Program control bits are memory mapped at
                                     ;FFCXH

      Prog_Mode=Prog_Mode AND 0000_0011B

      IF Prog_Mode=1 THEN GOTO Run_Mode
      IF Prog_Mode=2 THEN GOTO Key_Mode
Goto Macro_Loop


END ;{Main}
```

```
;///////////////////////////////////////////////////////////////////////////////
;**BASIC Subs****BASIC Subs****BASIC Subs****BASIC Subs****BASIC Subs****
;///////////////////////////////////////////////////////////////////////////////
;                        Basic Subroutines

;***************************************************************
Clear_Display:

;Clear RS
#ASM
          CLR P1.6    ;Reset Pin 1.6 low
#ASM_END


     ;Wait for 20 ms after power on
     msec_delay_value=20
     GOSUB Msec_Delay


     ;Send 30H and wait 6 ms
     XBY(0FFD0H)=030H : GOSUB Strobe_EN : msec_delay_value=6
     GOSUB Msec_Delay

     ;Send 30H and wait 1 ms
     XBY(0FFD0H)=030H : GOSUB Strobe_EN : msec_delay_value=1
     GOSUB Msec_Delay

     ;Send 30H and wait 1 ms
     XBY(0FFD0H)=030H : GOSUB Strobe_EN : msec_delay_value=1
     GOSUB Msec_Delay

     ;Function Set Send 0011_1000B and wait 1 ms
     XBY(0FFD0H)=0011_1000B : GOSUB Strobe_EN : msec_delay_value=1
     GOSUB Msec_Delay

     ;Display Off Send 0000_1100B and wait 1 ms
     XBY(0FFD0H)=0000_1100B : GOSUB Strobe_EN : msec_delay_value=1
     GOSUB Msec_Delay

     ;Display On Send 0000_0001B and wait 1 ms
     XBY(0FFD0H)=0000_0001B : GOSUB Strobe_EN : msec_delay_value=1
     GOSUB Msec_Delay

     ;Entry Mode Set Send 0000_0110B and wait 1 ms
     XBY(0FFD0H)=0000_0110B : GOSUB Strobe_EN : msec_delay_value=1
     GOSUB Msec_Delay

;Set RS
#ASM
          SETB P1.6   ;Set Pin 1.6 High
#ASM_END

Return ; {Clear_Display}
;***************************************************************
```

```
;**************************************************************
Disp_Mac_Len:
;This subroutine gets the value of the current recorded Macro
;length stored in AB_Length (IRAM address 01EH) and displays
;the value in bars and asterisks on the LCD.  Each byte in the
;Macro is represented by an asterisk on line 2. When line 2
;is full (16 asterisks) a bar is added to the string displayed on
;line 1, and line 2 is erased and starts over with one asterisk.
;the display can represent from 0 and 126 bytes.

      ;Get the current Macro length
      B_AB_Length=DBY(01EH)

      ;Calculate the number of bars to display
      IF B_AB_Length=0 THEN Num_Bars=0
      IF B_AB_Length>0 THEN Num_Bars=(B_AB_Length-1)/16


      ;Generate the Bar String (Line 1)
      STRtoLCD1$="RECORD: "
      FOR ii=1 TO 7
            ;Add a bar (Code FFH for the display), if needed
            IF Num_Bars>=ii THEN ASC(STRtoLCD1$,ii+7)=0FFH
            ;Add a period (Code 2EH for the display), if needed
            IF Num_Bars<ii THEN ASC(STRtoLCD1$,ii+7)=02EH
      NEXT ii
      ;Add a space (Code 20H for the display) to the end
      ASC(STRtoLCD1$,15)=020H

      ;Calculate the number of asterisks
      Num_Asterisks=B_AB_Length-(Num_Bars*16)

      ;Generate the Asterisk String (Line 2)
      STRtoLCD2$=""
      FOR ii=1 TO 16
            ;Add an asterisk if needed
            IF Num_Asterisks>=ii THEN STRtoLCD2$=STRtoLCD2$+"*"
            ;Add a space, if needed
            IF Num_Asterisks<ii THEN STRtoLCD2$=STRtoLCD2$+" "
      NEXT ii

      IF B_AB_Length>=126 THEN STRtoLCD1$="RECORD: XXXXXXX "
      IF B_AB_Length>=126 THEN STRtoLCD2$="   Macro Full    "

Return ; {Disp_Macro_Length}
;**************************************************************
```

```
;***************************************************************
Strobe_EN:

;This routine toggles P1.7 high then low, to indicate the
;data on latched on the 574 should be read by the LCD

#ASM
            SETB P1.7    ;Set Pin P1.7 high
            CLR P1.7     ;Reset Pin 1.7 low
#ASM_END

Return      ;{Strobe EN}
;***************************************************************

;***************************************************************
Msec_Delay:

;**** This routine delays the program for n milliseconds
;where n is the integer value stored in the variable
;msec_delay_value.  This routine has been specifically
;designed for the Dallas 320 33 MHz processor.  The timing
;uses program instruction cycle counting, it WILL NOT work
;correctly for any other processor/clock speed
;*Note: This delay is only approximate.  It should be used
;when a specific time needs to pass before another instruction
;is executed.  It should not be used for precise timing applications

;****
;This routine was "fine tuned" using empirical measurements with an
;oscilliscope

;if the wait is zero, return immediately
IF msec_delay_value=0 THEN RETURN

;Move delay value into Register 0
      dbY(00H)=msec_delay_value

#ASM
            mov R1,#255 ;move the values for 1 msec delay into R1
            mov R2,#10  ;and R2
Msec_Loop:  djnz R1,Msec_Loop
            mov R1,#255
            djnz R2,Msec_Loop
            mov R2,#10
            djnz R0,Msec_Loop

#ASM_END

RETURN ; {Msec_Delay}
;***************************************************************
```

```
 ;**************************************************************
 String_To_LCD1:

 ;This subroutine places the string value (up to 16 charaters)
 ;contained in STRtoLCD1$ onto the first line of the LCD Display

 ;Note cursor map is as follows
 ;
 ;
 ;Display      0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15   16 ...
 ;          ------------------------------------------------------------
 ;Line1      | 00 01 02 03 04 05 06 07 08 09  10  11  12  13  14  15 | 16 ...
 ;Line2      | 40 41 42 43 44 45 46 47 48 49  50  51  52  53  54  55 | 56 ...
 ;          ------------------------------------------------------------
 ;

 ;********** Place Cursor at Position 0, line 1, (home) prepare to Send data
#ASM
           CLR P1.6    ;Clear RS
#ASM_END

      XBY(0FFD0H)=080H : GOSUB Strobe_EN : msec_delay_value=1 : GOSUB Msec_Delay

#ASM
           SETB P1.6   ;Set RS
#ASM_END


      ;********** Loop through STRtoLCD1$ one character at a time
      ;Sending each ASCII character to the display.

      for j=0 to 15
           chaTemp=ASC(STRtoLCD1$,j)
           if chaTemp=0 then chaTemp=020H
           XBY(0FFD0H)=chaTemp : GOSUB Strobe_EN : msec_delay_value=1
           GOSUB Msec_Delay
      next j

RETURN ; {String_To_LCD1}
;**************************************************************

;**************************************************************
String_To_LCD2:

;This subroutine places the string value (up to 16 charaters)
;contained in STRtoLCD2$ onto the second line of the LCD Display

;********** Place Cursor at Position 40, line 1, prepare to Send data
#ASM
           CLR P1.6    ;Clear RS
#ASM_END

      XBY(0FFD0H)=0A8H : GOSUB Strobe_EN : msec_delay_value=1
      GOSUB Msec_Delay
```

```
#ASM
            SETB P1.6    ;Set RS
#ASM_END

      ;********** Loop through STRtoLCD2$ one character at a time
      ;Sending each ASCII character to the display.

      for j=0 to 15
            chaTemp=ASC(STRtoLCD2$,j)
            if chaTemp=0 then chaTemp=020H
            XBY(0FFD0H)=chaTemp : GOSUB Strobe_EN : msec_delay_value=1
            GOSUB Msec_Delay
      next j

RETURN ; {String_To_LCD2}
;***************************************************************


;***************************************************************
Enable_INT0:
;INT0 is used for Matrix_CLK
;This routine enables interrupt 0 to be triggered on falling edge
;(high to low) transitions.  The Interrupt Enable (IE) and Timer Control
;(TCON) Registers are masked to prevent interferring with other functions

      ;Make INT0 High-to-Low Edge Triggered
      TCON = TCON OR 0000_0001B

      ;Enable INT0 pin interrupt
      IE= IE OR 1000_0001B

RETURN ; {Enable_INT0}
;***************************************************************

;***************************************************************
Disable_INT0:
;INT0 is used for Matrix_CLK
;This routine disables interrupt 0

      ;Disable INT0 pin interrupt
      IE= IE AND 1111_1110B

RETURN ; {Disable_INT0}
;***************************************************************
```

```
;*************************************************************
Enable_INT1:
;INT1 is used for Keyi_CLK
;This routine enables interrupt 1 to be triggered on falling edge
;(high to low) transitions.  The Interrupt Enable (IE) and Timer Control
;(TCON) Registers are masked to prevent interferring with other functions

      ;Make INT1 High-to-Low Edge Triggered
      TCON = TCON OR 0000_0100B

      ;Enable INT1 pin interrupt
      IE= IE OR 1000_0100B

RETURN ; {Enable_INT1}
;*************************************************************

;*************************************************************
Disable_INT1:
;INT1 is used for Keyi_CLK
;This routine disables interrupt 1

      ;Disable INT1 pin interrupt
      IE= IE AND 1111_1011B

RETURN ; {Disable_INT1}
;*************************************************************


;*************************************************************
System_Reset:

;This routine clears any information stored in each of the
;four data sets, and sets the data set label to "Empty"
;This routine primarily is used after a new program is loaded, and is
;triggered from run mode by bringing the TX1 (P1.3) LO.
;There is no access for the user to this funtion, the
;reset is called from the HSMK board test button connected to TX1

      ;Display "System Reset"
      StrtoLCD1$="  System Reset  "
      GOSUB String_To_LCD1
      StrtoLCD2$="                "
      GOSUB String_To_LCD2

      ;Initialize Current Data Set to 0
      DBY(02EH)=0H

      ;Initialize Data Sets 1 through 4
      ;Clear the Lookup Table memory from 04000H to 0DFFFH
      FOR i=04000H to 0DFFFH
            XBY(i)=00H
      NEXT i
      ;Initialize the Data Set Labels
      GOSUB Empty_lblDS1
      GOSUB Empty_lblDS2
      GOSUB Empty_lblDS3
      GOSUB Empty_lblDS4

      GOTO Main_Startup

;{System_Reset}
;*************************************************************

;///////////////////////////////////////////////////////////////////////////////
;**ASM Subs****ASM Subs****ASM Subs****ASM Subs****ASM Subs****ASM Subs****
;///////////////////////////////////////////////////////////////////////////////
;                          Assembly Subroutines


#ASM
;*************************************************************
SB_Loop:
      ;This routine is used during both the run mode
      ;and the Macro Programming mode.  It passes any
      ;bytes loaded in the Send Buffer to the cpu.
      ;During run mode, Key presses from the Keyboard and the
      ;Matrix can result in data being loaded to the Send Buffer.
      ;During the Macro mode, only Key presses from the Keyboard are
      ;transferred to the Send Buffer.
      ;Send the bytes in the Send Buffer to the CPU
      ;after checking for traffic. Data traffic is detected
      ;by INTO, INT1, and checking the CPU_CLK.  Eech time any of these
      ;occur, they reset the traffic delay
      ;counter.  This counter is used to verify that no
      ;traffic has travelled on the lines for a certain period of
      ;time.  The Sending Flags are used to verify that there is
      ;no current traffic on the lines.

      MOV A,SB_Current                ;Move current Send Buffer position into A
      CJNE A,SB_Last,SB_Redlight      ;Compare current w/ last position
LJMP_SB_End:
      LJMP SB_End               ;If current and last are equal
                                ;there are no bytes to Send

      ;If there are bytes to Send, wait until there is no traffic detected
      ;on the CPU and Keyboard CLK and Data lines

SB_Redlight:
      JB Matrix_Sending,LJMP_SB_End ;jump out if Matrix is busy
      JB Key_Sending,LJMP_SB_End    ;jump out if Keyboard is busy
      JNB CPUi_CLK,LJMP_SB_End      ;jump out if CPU is busy
      MOV A,Traffic_Delay           ;Move the delay to the accumulator
      JZ SB_Greenlight              ;jump out of the loop if traffic delay=0
      DEC Traffic_Delay             ;subtract 1 from the traffic delay
      SJMP SB_Redlight
```

```
SB_Greenlight:
      ;Disable INT0 and INT1
      ANL IE,#11111010B        ;Clear Bit 0 and 2 in IE to disable INT0 and INT1

      ;Set Matrix and Keyboard CLK LO to
      ;Prevent them from Sending more data during a byte Send
      CLR OMatrixo_CLK
      CLR OKeyo_CLK

      INC SB_Current           ;get next byte in the Send Buffer

SB_ReSend:
      MOV DPL,SB_Current
      MOV DPH,SB_DPH
      MOVX A,@DPTR
      MOV B,A                  ;store the value in B for later

      MOV C,PSW.0              ;Get the Parity Bit for later
      CPL C                    ;Convert the bit to odd parity
      MOV Send_Parity,C        ;store the parity bit
      MOV R2,#8H               ;Use R2 as a counter for the number of data bits
                               ;sent

      JNB CPUi_CLK,SB_ReSend   ;Check CPU CLK Status

      ;-------------- Send Start Bit -----------------
      CLR OCPUo_Data           ;Bring Data LO

      MOV R7,#30               ;Delay
      LCALL Bit_Delay          ;

      CLR OCPUo_CLK            ;Bring CLK LO

      MOV R7,#60               ;Delay
      LCALL Bit_Delay          ;

      SETB OCPUo_CLK           ;Release CLK HI

      MOV R7,#30               ;Delay
      LCALL Bit_Delay          ;

      JNB CPUi_CLK,SB_ReSend   ;Check CPU CLK Status

            ;-------------- Send Data bits 0-7 ------------
SB_More_Bits:
      RRC A                    ;move the LSB to carry flag
      MOV OCPUo_Data,C         ;move the bit to the Data line

      MOV R7,#30               ;Delay
      LCALL Bit_Delay          ;

      CLR OCPUo_CLK            ;Make CLK LO

      MOV R7,#60               ;Delay
      LCALL Bit_Delay          ;
```

```
SETB OCPUo_CLK          ;set CLK HI

MOV R7,#30              ;Delay
LCALL Bit_Delay         ;

JNB CPUi_CLK,SB_ReSend  ;Check CPU CLK Status

DEC R2                        ;Subtract one from the number of data bits
                              ;left
CJNE R2,#0H,SB_More_Bits      ;Check to see if the last data bit was sent


;-------------- Send Parity Bit ---------------
MOV C,Send_Parity       ;get the stored parity bit
MOV OCPUo_Data,C        ;move the bit to the Data line

MOV R7,#30              ;Delay
LCALL Bit_Delay         ;

CLR OCPUo_CLK           ;Make CLK LO

MOV R7,#60              ;Delay
LCALL Bit_Delay         ;

SETB OCPUo_CLK          ;set CLK HI

MOV R7,#30              ;Delay
LCALL Bit_Delay         ;

JNB CPUi_CLK,SB_ReSend  ;Check CPU CLK Status

;-------------- Send Stop Bit ---------------
SETB OCPUo_Data         ;Release Stop Bit HI

MOV R7,#30              ;Delay
LCALL Bit_Delay         ;

CLR OCPUo_CLK           ;Set CLK LO

MOV R7,#60              ;Delay
LCALL Bit_Delay         ;

SETB OCPUo_CLK          ;Release CLK HI

MOV R7,#30              ;Delay
LCALL Bit_Delay         ;
```

```
SB_Wait_CPU:
        JNB CPUi_CLK,SB_Wait_CPU        ;Wait for CPU ACK


        ;Check to see if the byte just sent was a Caps Lock, Num Lock, or
        ;Scroll Lock Key press.  If it was, jump to the RM_LED routine
        ;immediately after the byte is sent to allow
        ;communication between the CPU and Keyboard

;***** CAPS LOCK
SB_CAPS:
        MOV A,B             ;Reload value into ACC
        CJNE A,#58H,SB_NUM              ;Check current byte for CAPS scan code (58H)
        JB CAPS_Down,CAPS_ChkLast       ;check for CAPS_Down (Key already pressed)
        LCALL LED                       ;If CAPS_Down=0, then call LED Subroutine
                                        ;to allow CPU to Keyboard Communication
                                        ;and change the LED status on the Keyboard
        SETB CAPS_Down                  ;Set the CAPS_Down status = 1 (pressed)
        AJMP SB_NUM                     ;Jump out of loop

CAPS_ChkLast:
        MOV A,SB_Prev_Byte              ;Get the previous byte sent from SB
        CJNE A,#0F0H,SB_NUM             ;See if previous byte was F0H (release code)
        CLR CAPS_Down                   ;if it was, set CAPS_Down = 0 (released)

;***** NUM LOCK
SB_NUM:
        MOV A,B                         ;Reload value into ACC
        CJNE A,#77H,SB_SCROLL           ;Check current byte for NUM scan code (77H)
        JB NUM_Down,NUM_ChkLast         ;check for NUM_Down (Key already pressed)
        LCALL LED                       ;If NUM_Down=0, then call LED Subroutine
                                        ;to allow CPU to Keyboard Communication
                                        ;and change the LED status on the Keyboard
        SETB NUM_Down                   ;Set the NUM_Down status = 1 (pressed)
        AJMP SB_SCROLL                  ;Jump out of loop

NUM_ChkLast:
        MOV A,SB_Prev_Byte                    ;Get the previous byte sent from SB
        CJNE A,#0F0H,SB_SCROLL          ;See if previous byte was F0H (release code)
        CLR NUM_Down                    ;if it was, set NUM_Down = 0 (released)


;***** SCROLL LOCK
SB_SCROLL:
        MOV A,B                         ;Reload value into ACC
        CJNE A,#7EH,SB_End_Lock         ;Check current byte for SCROLL scan code
                                        ;(7EH)
        JB SCROLL_Down,SCROLL_ChkLast   ;check for SCROLL_Down (Key already pressed)
        LCALL LED                       ;If SCROLL_Down=0, then call LED Subroutine
                                        ;to allow CPU to Keyboard Communication
                                        ;and change the LED status on the Keyboard
        SETB SCROLL_Down                ;Set the SCROLL_Down status = 1 (pressed)
        AJMP SB_End_Lock                ;Jump out of loop
```

```
SCROLL_ChkLast:
      MOV A,SB_Prev_Byte             ;Get the previous byte sent from SB
      CJNE A,#0F0H,SB_End_Lock       ;See if previous byte was F0H (release code)
      CLR SCROLL_Down                ;if it was, set SCROLL_Down = 0 (released)


SB_End_Lock:
      MOV SB_Prev_Byte,B

      ;Clear the current byte in the Send Buffer (the byte just sent)
SB_Clear_Byte:
      MOV A,#00H
      MOVX @DPTR,A

      ;Reset Matrix byte variables
      ;This is to make sure no bits were transferred during
      ;the short time between when no Matrix traffic was detected
      ;and the Matrix interrupt was disabled (INT0)
      ;If this does occur, the Matrix will re-send the aborted byte
      ;once the Matrix CLK line is brought back high.
      CLR Matrix_Sending
      MOV A,#00H
      MOV Matrix_Bit_Num,A
      MOV Matrix_Byte,A

      ;Reset Key byte Variables
      ;This is to make sure no bits were transferred during
      ;the short time between when no Keyboard traffic was detected
      ;and the Keyboard interrupt was disabled (INT0)
      ;If this does occur, the Matrix will re-send the aborted byte
      ;once the Keyboard CLK line is brought back high.
      CLR Key_Sending
      MOV A,#00H
      MOV Key_Bit_Num,A
      MOV Key_Byte,A

      ;Clear the Interrupt Flags
      ANL 088H,#11110101B      ;Clear INT0,INT1 Flag

      ;Enable the Interrupts
      ORL IE,#00000101B ;Set Bit 0 and 2 in IE to enable INT0 and INT1


      ;Release Matrix and Keyboard CLK to
      ;allow traffic again
      SETB OKeyo_CLK           ;Release Keyboard
      SETB OMatrixo_CLK        ;Release Matrix

      MOV A,#20H
      MOV Traffic_Delay,A            ;Put delay in Traffic_Delay

SB_End:              ;End of the Send Buffer Routine

      RET            ;Return from subroutine
;***************************************************************
#ASM_END
```

```
#ASM
;*****************************************************************
;This subroutine provides the necessary delay between
;bits sent to the CPU
;Uses the value in R7 when called to delay the processor
Bit_Delay:
      DEC R7
      CJNE R7,#0H,Bit_Delay
      RET  ;{Bit_Delay}
;*****************************************************************
#ASM_END

#ASM
;*****************************************************************
;This subroutine provides the communication protocol between the
;CPU and the Keyboard when a LOCK Key is depressed and sent from the
;Send Buffer
LED:

      ;******* CPU To Keyboard (EDH)
      LCALL CPUtoKeyboard

      ;******* Keyboard to CPU (FAH)
      LCALL KeyboardtoCPU

      ;******* CPU To Keyboard (XX)
      LCALL CPUtoKeyboard

      ;******* Keyboad to CPU (FAH)
      LCALL KeyboardtoCPU

      ;******* Restore the variables
      CLR OKeyo_CLK

      RET ;{LED}


;--------------------
CPUtoKeyboard:

      ;Bring Key CLK LO
      CLR OKeyo_CLK

      ;Wait for CPU Data to go LO
LED_Wait1:
      JB CPUi_Data,LED_Wait1

      ;Send the data to the Keyboard
      ;Store 0AH in Accumulator
      ;This will count the bits sent
      MOV A,#0AH

      ;Bring the Key Data Line LO
      CLR OKeyo_Data
```

```
     ;Wait for the CPU CLK to go HI
LED_Wait2:
     JNB CPUi_CLK,LED_Wait2

     ;Release Key CLK Line
     SETB OKeyo_CLK

     ;Delay for setup
     MOV R7,#06H
     LCALL Bit_Delay

     ;Wait for Key CLK to go LO
LED_Wait3:
     JB Keyi_CLK,LED_Wait3

     ;Set CPU CLK LO
     CLR OCPUo_CLK

     ;delay for CPU data to set up
     MOV R7,#020H
     LCALL Bit_Delay

     ;Transfer Data from CPU to KEY
     MOV C,CPUi_Data
     MOV OKeyo_Data,C

     ;Wait for Key CLK to go HI
LED_Wait4:
     JNB Keyi_CLK,LED_Wait4

     ;Release CPU CLK
     SETB OCPUo_CLK


     ;Decrement the Accumulator
     ;Jump to the Acknowledge section if
     ;Accumulator=0
     DEC ACC
     JZ LED_ACK1

     LJMP LED_Wait3

     ;Transfer ACK Signal from Keyboard
LED_ACK1:

     ;Wait for Key CLK to go LO
LED_Wait5:
     JB Keyi_CLK,LED_Wait5

     ;Pull CPU Data LO
     CLR OCPUo_Data

     ;delay for data to set up
     MOV R7,#03H
     LCALL Bit_Delay
```

```
      ;Pull CPU CLK LO
      CLR OCPUo_CLK

      ;Wait for Key CLK to go HI
LED_Wait6:
      JNB Keyi_CLK,LED_Wait6

      ;Release CPU CLK
      SETB OCPUo_CLK

      ;Release CPU Data
      SETB OCPUo_Data

      RET ;{CPUtoKeyboard}

;--------------------

;--------------------
KeyboardtoCPU:

      ;Use accumulator to count the number of bits sent
      MOV A,#0BH

      ;Wait for Key CLK to go LO
LED_Wait7:
      JB Keyi_CLK,LED_Wait7

      ;Get the Keyboard Data value
      MOV C,Keyi_Data

      ;Put the Keyboard data value on the CPU Data line
      MOV OCPUo_Data,C

      ;Delay for setup
      MOV R7,#03H
      LCALL Bit_Delay

      ;Bring the CPU Clock line LO
      CLR OCPUo_CLK

      ;Wait until the Keyboard Clock goes HI
LED_Wait8:
      JNB Keyi_CLK,LED_Wait8

      ;Bring the CPU Clock Line back HI
      SETB OCPUo_CLK

      ;Decrement the Accumulator
      ;Jump to the Acknowledge section if
      ;Accumulator=0
      DEC ACC
      JZ LED_ACK2
      AJMP LED_Wait7

LED_ACK2:
      ;Wait for CPU CLK to go LO
LED_Wait9:
      JB CPUi_CLK,LED_Wait9

      ;Pull Key CLK LO
      CLR OKeyo_CLK

      ;Wait for CPU CLK to go HI
LED_Wait10:
      JNB CPUi_CLK,LED_Wait10

      ;Release Key CLK
      SETB OKeyo_CLK

      RET ;{KeyboardtoCPU}
;--------------------

;******************************************************************
#ASM_END



;////////////////////////////////////////////////////////////////////////////////
;**INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0**
;////////////////////////////////////////////////////////////////////////////////
;                         Interrupt 0 Vector

;Interrupt Service Routine 0
;
;This routine services the INT0.
;The ISR location is jumped to from the INT0 Vector at Code Address 0003H
;The Program Status Word is Pushed onto the stack followed by a
;Long Jump to the Full ISR0.  The actual location of the ISR in
;code space is handled by the compiler.

#ASM

Coder0 set $      ;Save the program counter
                  ;$ is the current program counter value
                  ;and is two-bytes long

      ORG 0003H   ;Initialize the INT0 Vector

      PUSH PSW    ;Save the status flags

      LJMP _ISR0  ;long jump to the INT0 ISR Location

      ORG Coder0  ;Restore the program counter to allow the compiler
                  ;to use the code space efficiently
```

```
;/////////////////////////////////////////////////////////////////////////////////
;**INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0**
;/////////////////////////////////////////////////////////////////////////////////
;                             Interrupt 0 Startup

_ISR0:                 ;This is the Interrupt Service Routine for Interrupt 0
       PUSH ACC
       PUSH DPL
       PUSH DPH

       SETB Matrix_Sending


;/////////////////////////////////////////////////////////////////////////////////
;**INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0**
;/////////////////////////////////////////////////////////////////////////////////
;                             Interrupt 0 Main


       ;Get the current bit status right away
       ;(this makes sure you do not miss the data)
       MOV C,Matrixi_Data      ;get the Data bit in the carry flag

       MOV Matrix_Bit,C        ;Save the Data bit

       MOV A,Matrix_Bit_Num    ;Get the number of bits already read
       JZ Zero_Next_Bit  ;If this is the first bit (Bit Number = 0)
                               ;jump out and increment the bit number

       DEC A
       DEC A
       DEC A
       DEC A
       DEC A
       DEC A
       DEC A
       DEC A
       DEC A
       JZ Zero_Next_Bit  ;also jump out if BIT_NUMBER = 9

       DEC A
       JZ Zero_Last_Bit        ;if this is the last bit (#10), update stuff

       ;do this stuff for bits 1 through 8
       MOV A,Matrix_Byte       ;Get the current byte
       MOV C,Matrix_Bit        ;Move the current bit into the carry flag
       RRC A                   ;Rotate the current bit into position (LSB)
       MOV Matrix_Byte,A       ;Save the current byte


Zero_Next_Bit:
       INC Matrix_Bit_Num      ;Add one to the current bit number
       AJMP INT0_Exit
```

```
Zero_Last_Bit:
      ;put the received byte into the Matrix Buffer
      INC MB_Last
      MOV A,MB_DPH              ;Get the high byte pointer to the Matrix Buffer
      MOV DPH,A
      MOV A,MB_Last             ;Get the low byte pointer to the Matrix Buffer
      MOV DPL,A
      MOV A,Matrix_Byte         ;Get the byte just received
      MOVX @DPTR,A              ;Store the byte in the Matrix Buffer

      ;Clear the INT0 variables
      MOV A,#0H
      MOV Matrix_Bit_Num,A      ;Reset the Bit Number
      MOV Matrix_Byte,A         ;Reset Current Byte to 00H
      CLR Matrix_Sending        ;Clear the Sending flag

;///////////////////////////////////////////////////////////////////////////////
;**INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0****INT 0**
;///////////////////////////////////////////////////////////////////////////////
;                         Interrupt 0 Exit

INT0_Exit:
      MOV A,#20H
      MOV Traffic_Delay,A             ;Put delay in Traffic_Delay

      POP DPH
      POP DPL
      POP ACC               ;Restore the Accumulator
      POP PSW               ;Restore the Program Status Word

      RETI                  ;Return from INT0

#ASM_END


;///////////////////////////////////////////////////////////////////////////////
;**INT 1****INT 1****INT 1****INT 1****INT 1****INT 1****INT 1****INT 1**
;///////////////////////////////////////////////////////////////////////////////
;                         Interrupt 1 Vector

;Interrupt Service Routine 1
;
;This routine services the INT1.
;ISR1 is called when the Keyboard brings the Clock line LO.  This is done when
;the Keyboard Sends data to the CPU.
;The ISR location is jumped to from the INT1 Vector at Code Address 0013H
;The Program Status Word is Pushed onto the stack followed by a
;Long Jump to the Full ISR1.  The actual location of the ISR in
;code space is handled by the compiler.
```

```
#ASM

Code_Ret1 set $     ;Save the program counter
                    ;$ is the current program counter value
                    ;and is two-bytes long

     ORG 0013H      ;Initialize the INT1 Vector

     PUSH PSW       ;Save the status flags

     LJMP _ISR1     ;long jump to the INT1 ISR Location

     ORG Code_Ret1       ;Restore the program counter to allow the compiler
                    ;to use the code space efficiently


;///////////////////////////////////////////////////////////////////////////////
;**INT 1****INT 1****INT 1****INT 1****INT 1****INT 1****INT 1****INT 1**
;///////////////////////////////////////////////////////////////////////////////
;                          Interrupt 1 Startup

_ISR1:              ;This is the Interrupt Service Routine for Interrupt 1
     PUSH ACC
     PUSH DPL
     PUSH DPH

     SETB Key_Sending

;///////////////////////////////////////////////////////////////////////////////
;**INT 1****INT 1****INT 1****INT 1****INT 1****INT 1****INT 1****INT 1**
;///////////////////////////////////////////////////////////////////////////////
;                          Interrupt 1 Main


     ;Get the current bit status right away
     ;(this makes sure you do not miss the data)
     MOV C,Keyi_Data          ;get the Data bit in the carry flag

     MOV Key_Bit,C            ;Save the Data bit

     MOV A,Key_Bit_Num        ;Get the number of bits already read
     JZ One_Next_Bit          ;If this is the first bit (Bit Number = 0)
                              ;jump out and increment the bit number

     DEC A
     DEC A
     DEC A
     DEC A
     DEC A
     DEC A
     DEC A
     DEC A
     DEC A
     JZ One_Next_Bit          ;also jump out if BIT_NUMBER = 9
```

```
        DEC A
        JZ One_Last_Bit           ;if this is the last bit (#10), update stuff

        ;do this stuff for bits 1 through 8
        MOV A,Key_Byte            ;Get the current byte
        MOV C,Key_Bit             ;Move the current bit into the carry flag
        RRC A                     ;Rotate the current bit into position (LSB)
        MOV Key_Byte,A            ;Save the current byte


One_Next_Bit:
        INC Key_Bit_Num    ;Add one to the current bit number
        LJMP INT1_Exit


One_Last_Bit:
        ;put the received byte into the Key Buffer
        INC KB_Last

        MOV A,KB_DPH              ;Get the high byte pointer to the Key Buffer
        MOV DPH,A
        MOV A,KB_Last             ;Get the low byte pointer to the Key Buffer
        MOV DPL,A
        MOV A,Key_Byte            ;Get the byte just received
        MOVX @DPTR,A              ;Store the byte in the Key Buffer

        ;Clear the INT1 variables
        MOV A,#0H
        MOV Key_Bit_Num,A         ;Reset the Bit Number
        MOV Key_Byte,A            ;Reset Current Byte to 00H
        CLR Key_Sending           ;Clear the Sending flag


;///////////////////////////////////////////////////////////////////////////////
;**INT 1****INT 1****INT 1****INT 1****INT 1****INT 1****INT 1****INT 1**
;///////////////////////////////////////////////////////////////////////////////

;                         Interrupt 1 Exit
INT1_Exit:
        MOV A,#20H
        MOV Traffic_Delay,A       ;Put delay in Traffic_Delay

        POP DPH
        POP DPL
        POP ACC                   ;Restore the Accumulator
        POP PSW                   ;Restore the Program Status Word

        RETI                      ;Return from INT1

#ASM_END
```

```
;///////////////////////////////////////////////////////////////////////////////
;*** Lookup Tables *** Lookup Tables *** Lookup Tables *** Lookup Tables ***
;///////////////////////////////////////////////////////////////////////////////

;The following table is used to calculate the location of the 128 bytes assigned
;to a particular Matrix Pad.  Each of the 80 Pads in the Matrix is represented
;byte a one byte code (XXH for Key down, F0H XXH for Key up.)  The software uses
;the Matrix byte XXH to calculate the base storage location for the bytes bound
;to the Pad. Base Address = 4000H + [(Pad Number from Lookup Table )* 7FH ]
;The current data set offset (Set0=+0, Set1=+2800H, Set2=+5000H, Set3=+7800) is
;added to the base address to get the memory storage location..

;Current Location = Base Address + Offset Address


#ASM

Pad_Lookup:
        db 0FFH      ;Matrix Data Code 0, Not Used
        db 0FFH      ;Matrix Data Code 1, Not Used
        db 0FFH      ;Matrix Data Code 2, Not Used
        db 15        ;Matrix Data Code 3, -------- Matrix Pad 15
        db 75        ;Matrix Data Code 4, -------- Matrix Pad 75
        db 63        ;Matrix Data Code 5, -------- Matrix Pad 63
        db 7         ;Matrix Data Code 6, -------- Matrix Pad 7
        db 6         ;Matrix Data Code 7, -------- Matrix Pad 6
        db 0FFH      ;Matrix Data Code 8, Not Used
        db 64        ;Matrix Data Code 9, -------- Matrix Pad 64
        db 18        ;Matrix Data Code A, -------- Matrix Pad 18
        db 16        ;Matrix Data Code B, -------- Matrix Pad 16
        db 8         ;Matrix Data Code C, -------- Matrix Pad 8
        db 50        ;Matrix Data Code D, -------- Matrix Pad 50
        db 34        ;Matrix Data Code E, -------- Matrix Pad 34
        db 0FFH      ;Matrix Data Code F, Not Used

        db 0FFH      ;Matrix Data Code 10, Not Used
        db 37        ;Matrix Data Code 11, -------- Matrix Pad 37
        db 0FFH      ;Matrix Data Code 12, Not Used
        db 0FFH      ;Matrix Data Code 13, Not Used
        db 38        ;Matrix Data Code 14, -------- Matrix Pad 38
        db 29        ;Matrix Data Code 15, -------- Matrix Pad 29
        db 2         ;Matrix Data Code 16, -------- Matrix Pad 2
        db 0FFH      ;Matrix Data Code 17, Not Used
        db 0FFH      ;Matrix Data Code 18, Not Used
        db 0FFH      ;Matrix Data Code 19, Not Used
        db 51        ;Matrix Data Code 1A, -------- Matrix Pad 51
        db 30        ;Matrix Data Code 1B, -------- Matrix Pad 30
        db 42        ;Matrix Data Code 1C, -------- Matrix Pad 42
        db 60        ;Matrix Data Code 1D, -------- Matrix Pad 60
        db 3         ;Matrix Data Code 1E, -------- Matrix Pad 3
        db 0FFH      ;Matrix Data Code 1F, Not Used

        db 0FFH      ;Matrix Data Code 20, Not Used
        db 0FFH      ;Matrix Data Code 21, Not Used
        db 69        ;Matrix Data Code 22, -------- Matrix Pad 69
        db 52        ;Matrix Data Code 23, -------- Matrix Pad 52
        db 53        ;Matrix Data Code 24, -------- Matrix Pad 53
db 11         ;Matrix Data Code 25, -------- Matrix Pad 11
db 4          ;Matrix Data Code 26, -------- Matrix Pad 4
db 0FFH       ;Matrix Data Code 27, Not Used
db 0FFH       ;Matrix Data Code 28, Not Used
db 40         ;Matrix Data Code 29, -------- Matrix Pad 40
db 59         ;Matrix Data Code 2A, -------- Matrix Pad 59
db 62         ;Matrix Data Code 2B, -------- Matrix Pad 62
db 49         ;Matrix Data Code 2C, -------- Matrix Pad 49
db 0          ;Matrix Data Code 2D, -------- Matrix Pad 0
db 12         ;Matrix Data Code 2E, -------- Matrix Pad 12
db 0FFH       ;Matrix Data Code 2F, Not Used

db 0FFH       ;Matrix Data Code 30, Not Used
db 45         ;Matrix Data Code 31, -------- Matrix Pad 45
db 43         ;Matrix Data Code 32, -------- Matrix Pad 43
db 27         ;Matrix Data Code 33, -------- Matrix Pad 27
db 26         ;Matrix Data Code 34, -------- Matrix Pad 26
db 70         ;Matrix Data Code 35, -------- Matrix Pad 70
db 13         ;Matrix Data Code 36, -------- Matrix Pad 13
db 0FFH       ;Matrix Data Code 37, Not Used
db 0FFH       ;Matrix Data Code 38, Not Used
db 0FFH       ;Matrix Data Code 39, Not Used
db 78         ;Matrix Data Code 3A, -------- Matrix Pad 78
db 28         ;Matrix Data Code 3B, -------- Matrix Pad 28
db 77         ;Matrix Data Code 3C, -------- Matrix Pad 77
db 14         ;Matrix Data Code 3D, -------- Matrix Pad 14
db 21         ;Matrix Data Code 3E, -------- Matrix Pad 21
db 0FFH       ;Matrix Data Code 3F, Not Used

db 0FFH       ;Matrix Data Code 40, Not Used
db 0FFH       ;Matrix Data Code 41, Not Used
db 35         ;Matrix Data Code 42, -------- Matrix Pad 35
db 76         ;Matrix Data Code 43, -------- Matrix Pad 76
db 46         ;Matrix Data Code 44, -------- Matrix Pad 46
db 1          ;Matrix Data Code 45, -------- Matrix Pad 1
db 71         ;Matrix Data Code 46, -------- Matrix Pad 71
db 0FFH       ;Matrix Data Code 47, Not Used
db 0FFH       ;Matrix Data Code 48, Not Used
db 24         ;Matrix Data Code 49, -------- Matrix Pad 24
db 31         ;Matrix Data Code 4A, -------- Matrix Pad 31
db 36         ;Matrix Data Code 4B, -------- Matrix Pad 36
db 73         ;Matrix Data Code 4C, -------- Matrix Pad 73
db 47         ;Matrix Data Code 4D, -------- Matrix Pad 47
db 72         ;Matrix Data Code 4E, -------- Matrix Pad 72
db 0FFH       ;Matrix Data Code 4F, Not Used

db 0FFH       ;Matrix Data Code 50, Not Used
db 0FFH       ;Matrix Data Code 51, Not Used
db 22         ;Matrix Data Code 52, -------- Matrix Pad 22
db 0FFH       ;Matrix Data Code 53, Not Used
db 32         ;Matrix Data Code 54, -------- Matrix Pad 32
db 41         ;Matrix Data Code 55, -------- Matrix Pad 41
db 0FFH       ;Matrix Data Code 56, Not Used
db 0FFH       ;Matrix Data Code 57, Not Used
db 74         ;Matrix Data Code 58, -------- Matrix Pad 74
db 0FFH       ;Matrix Data Code 59, Not Used
db 54         ;Matrix Data Code 5A, -------- Matrix Pad 54
db 33       ;Matrix Data Code 5B, -------- Matrix Pad 33
db 0FFH     ;Matrix Data Code 5C, Not Used
db 23       ;Matrix Data Code 5D, -------- Matrix Pad 23
db 0FFH     ;Matrix Data Code 5E, Not Used
db 0FFH     ;Matrix Data Code 5F, Not Used

db 0FFH     ;Matrix Data Code 60, Not Used
db 0FFH     ;Matrix Data Code 61, Not Used
db 0FFH     ;Matrix Data Code 62, Not Used
db 0FFH     ;Matrix Data Code 63, Not Used
db 0FFH     ;Matrix Data Code 64, Not Used
db 0FFH     ;Matrix Data Code 65, Not Used
db 44       ;Matrix Data Code 66, -------- Matrix Pad 44
db 0FFH     ;Matrix Data Code 67, Not Used
db 0FFH     ;Matrix Data Code 68, Not Used
db 58       ;Matrix Data Code 69, -------- Matrix Pad 58
db 0FFH     ;Matrix Data Code 6A, Not Used
db 67       ;Matrix Data Code 6B, -------- Matrix Pad 67
db 10       ;Matrix Data Code 6C, -------- Matrix Pad 10
db 0FFH     ;Matrix Data Code 6D, Not Used
db 0FFH     ;Matrix Data Code 6E, Not Used
db 0FFH     ;Matrix Data Code 6F, Not Used

db 57       ;Matrix Data Code 70, -------- Matrix Pad 57
db 55       ;Matrix Data Code 71, -------- Matrix Pad 55
db 65       ;Matrix Data Code 72, -------- Matrix Pad 65
db 68       ;Matrix Data Code 73, -------- Matrix Pad 68
db 9        ;Matrix Data Code 74, -------- Matrix Pad 9
db 19       ;Matrix Data Code 75, -------- Matrix Pad 19
db 61       ;Matrix Data Code 76, -------- Matrix Pad 61
db 25       ;Matrix Data Code 77, -------- Matrix Pad 25
db 5        ;Matrix Data Code 78, -------- Matrix Pad 5
db 56       ;Matrix Data Code 79, -------- Matrix Pad 56
db 66       ;Matrix Data Code 7A, -------- Matrix Pad 66
db 48       ;Matrix Data Code 7B, -------- Matrix Pad 48
db 79       ;Matrix Data Code 7C, -------- Matrix Pad 79
db 20       ;Matrix Data Code 7D, -------- Matrix Pad 20
db 39       ;Matrix Data Code 7E, -------- Matrix Pad 39
db 0FFH     ;Matrix Data Code 7F, Not Used

db 0FFH     ;Matrix Data Code 80, Not Used
db 0FFH     ;Matrix Data Code 81, Not Used
db 0FFH     ;Matrix Data Code 82, Not Used
db 17       ;Matrix Data Code 83, -------- Matrix Pad 17
db 0FFH     ;Matrix Data Code 84, Not Used
db 0FFH     ;Matrix Data Code 85, Not Used
db 0FFH     ;Matrix Data Code 86, Not Used
db 0FFH     ;Matrix Data Code 87, Not Used
db 0FFH     ;Matrix Data Code 88, Not Used
db 0FFH     ;Matrix Data Code 89, Not Used
db 0FFH     ;Matrix Data Code 8A, Not Used
db 0FFH     ;Matrix Data Code 8B, Not Used
db 0FFH     ;Matrix Data Code 8C, Not Used
db 0FFH     ;Matrix Data Code 8D, Not Used
db 0FFH     ;Matrix Data Code 8E, Not Used
db 0FFH     ;Matrix Data Code 8F, Not Used
db 0FFH       ;Matrix Data Code 90, Not Used
db 0FFH       ;Matrix Data Code 91, Not Used
db 0FFH       ;Matrix Data Code 92, Not Used
db 0FFH       ;Matrix Data Code 93, Not Used
db 0FFH       ;Matrix Data Code 94, Not Used
db 0FFH       ;Matrix Data Code 95, Not Used
db 0FFH       ;Matrix Data Code 96, Not Used
db 0FFH       ;Matrix Data Code 97, Not Used
db 0FFH       ;Matrix Data Code 98, Not Used
db 0FFH       ;Matrix Data Code 99, Not Used
db 0FFH       ;Matrix Data Code 9A, Not Used
db 0FFH       ;Matrix Data Code 9B, Not Used
db 0FFH       ;Matrix Data Code 9C, Not Used
db 0FFH       ;Matrix Data Code 9D, Not Used
db 0FFH       ;Matrix Data Code 9E, Not Used
db 0FFH       ;Matrix Data Code 9F, Not Used

db 0FFH       ;Matrix Data Code A0, Not Used
db 0FFH       ;Matrix Data Code A1, Not Used
db 0FFH       ;Matrix Data Code A2, Not Used
db 0FFH       ;Matrix Data Code A3, Not Used
db 0FFH       ;Matrix Data Code A4, Not Used
db 0FFH       ;Matrix Data Code A5, Not Used
db 0FFH       ;Matrix Data Code A6, Not Used
db 0FFH       ;Matrix Data Code A7, Not Used
db 0FFH       ;Matrix Data Code A8, Not Used
db 0FFH       ;Matrix Data Code A9, Not Used
db 0FFH       ;Matrix Data Code AA, Not Used
db 0FFH       ;Matrix Data Code AB, Not Used
db 0FFH       ;Matrix Data Code AC, Not Used
db 0FFH       ;Matrix Data Code AD, Not Used
db 0FFH       ;Matrix Data Code AE, Not Used
db 0FFH       ;Matrix Data Code AF, Not Used

db 0FFH       ;Matrix Data Code B0, Not Used
db 0FFH       ;Matrix Data Code B1, Not Used
db 0FFH       ;Matrix Data Code B2, Not Used
db 0FFH       ;Matrix Data Code B3, Not Used
db 0FFH       ;Matrix Data Code B4, Not Used
db 0FFH       ;Matrix Data Code B5, Not Used
db 0FFH       ;Matrix Data Code B6, Not Used
db 0FFH       ;Matrix Data Code B7, Not Used
db 0FFH       ;Matrix Data Code B8, Not Used
db 0FFH       ;Matrix Data Code B9, Not Used
db 0FFH       ;Matrix Data Code BA, Not Used
db 0FFH       ;Matrix Data Code BB, Not Used
db 0FFH       ;Matrix Data Code BC, Not Used
db 0FFH       ;Matrix Data Code BD, Not Used
db 0FFH       ;Matrix Data Code BE, Not Used
db 0FFH       ;Matrix Data Code BF, Not Used

db 0FFH       ;Matrix Data Code C0, Not Used
db 0FFH       ;Matrix Data Code C1, Not Used
db 0FFH       ;Matrix Data Code C2, Not Used
db 0FFH       ;Matrix Data Code C3, Not Used
db 0FFH       ;Matrix Data Code C4, Not Used
db 0FFH       ;Matrix Data Code C5, Not Used
db 0FFH        ;Matrix Data Code C6, Not Used
db 0FFH        ;Matrix Data Code C7, Not Used
db 0FFH        ;Matrix Data Code C8, Not Used
db 0FFH        ;Matrix Data Code C9, Not Used
db 0FFH        ;Matrix Data Code CA, Not Used
db 0FFH        ;Matrix Data Code CB, Not Used
db 0FFH        ;Matrix Data Code CC, Not Used
db 0FFH        ;Matrix Data Code CD, Not Used
db 0FFH        ;Matrix Data Code CE, Not Used
db 0FFH        ;Matrix Data Code CF, Not Used

db 0FFH        ;Matrix Data Code D0, Not Used
db 0FFH        ;Matrix Data Code D1, Not Used
db 0FFH        ;Matrix Data Code D2, Not Used
db 0FFH        ;Matrix Data Code D3, Not Used
db 0FFH        ;Matrix Data Code D4, Not Used
db 0FFH        ;Matrix Data Code D5, Not Used
db 0FFH        ;Matrix Data Code D6, Not Used
db 0FFH        ;Matrix Data Code D7, Not Used
db 0FFH        ;Matrix Data Code D8, Not Used
db 0FFH        ;Matrix Data Code D9, Not Used
db 0FFH        ;Matrix Data Code DA, Not Used
db 0FFH        ;Matrix Data Code DB, Not Used
db 0FFH        ;Matrix Data Code DC, Not Used
db 0FFH        ;Matrix Data Code DD, Not Used
db 0FFH        ;Matrix Data Code DE, Not Used
db 0FFH        ;Matrix Data Code DF, Not Used

db 0FFH        ;Matrix Data Code E0, Not Used
db 0FFH        ;Matrix Data Code E1, Not Used
db 0FFH        ;Matrix Data Code E2, Not Used
db 0FFH        ;Matrix Data Code E3, Not Used
db 0FFH        ;Matrix Data Code E4, Not Used
db 0FFH        ;Matrix Data Code E5, Not Used
db 0FFH        ;Matrix Data Code E6, Not Used
db 0FFH        ;Matrix Data Code E7, Not Used
db 0FFH        ;Matrix Data Code E8, Not Used
db 0FFH        ;Matrix Data Code E9, Not Used
db 0FFH        ;Matrix Data Code EA, Not Used
db 0FFH        ;Matrix Data Code EB, Not Used
db 0FFH        ;Matrix Data Code EC, Not Used
db 0FFH        ;Matrix Data Code ED, Not Used
db 0FFH        ;Matrix Data Code EE, Not Used
db 0FFH        ;Matrix Data Code EF, Not Used

db 0FFH        ;Matrix Data Code F0, Not Used
db 0FFH        ;Matrix Data Code F1, Not Used
db 0FFH        ;Matrix Data Code F2, Not Used
db 0FFH        ;Matrix Data Code F3, Not Used
db 0FFH        ;Matrix Data Code F4, Not Used
db 0FFH        ;Matrix Data Code F5, Not Used
db 0FFH        ;Matrix Data Code F6, Not Used
db 0FFH        ;Matrix Data Code F7, Not Used
db 0FFH        ;Matrix Data Code F8, Not Used
db 0FFH        ;Matrix Data Code F9, Not Used
db 0FFH        ;Matrix Data Code FA, Not Used
db 0FFH        ;Matrix Data Code FB, Not Used
```

```
        db 0FFH      ;Matrix Data Code FC, Not Used
        db 0FFH      ;Matrix Data Code FD, Not Used
        db 0FFH      ;Matrix Data Code FE, Not Used
        db 0FFH      ;Matrix Data Code FF, Not Used


;The following Lookup Table is used to generate ASCII codes
;from certain Keys on the Keyboard, to display labels on
;the LCD.  While in the Label Mode, the user presses a Key
;on the Keyboard.  The code generated by the Key press
;is used in this lookup table to find the corresponding ASCII
;character for that Key.  In the Label Mode, The LCD only displays
;letters, numbers and some punctuation.  All letters are capitalized.
;Special Keys and Function Keys (other than backspace
;and enter) are ignored during the Label Mode.

Label_Lookup:
        db 0FFH      ;Matrix Data Code 0, Not Used
        db 0FFH      ;Matrix Data Code 1, Not Used
        db 0FFH      ;Matrix Data Code 2, Not Used
        db 0FFH      ;Matrix Data Code 3, Not Used
        db 0FFH      ;Matrix Data Code 4, Not Used
        db 0FFH      ;Matrix Data Code 5, Not Used
        db 0FFH      ;Matrix Data Code 6, Not Used
        db 0FFH      ;Matrix Data Code 7, Not Used
        db 0FFH      ;Matrix Data Code 8, Not Used
        db 0FFH      ;Matrix Data Code 9, Not Used
        db 0FFH      ;Matrix Data Code A, Not Used
        db 0FFH      ;Matrix Data Code B, Not Used
        db 0FFH      ;Matrix Data Code C, Not Used
        db 0FFH      ;Matrix Data Code D, Not Used
        db 0FFH      ;Matrix Data Code E, Not Used
        db 0FFH      ;Matrix Data Code F, Not Used

        db 0FFH      ;Matrix Data Code 10, Not Used
        db 0FFH      ;Matrix Data Code 11, Not Used
        db 0FFH      ;Matrix Data Code 12, Not Used
        db 0FFH      ;Matrix Data Code 13, Not Used
        db 0FFH      ;Matrix Data Code 14, Not Used
        db 'Q'       ;Matrix Data Code 15, ******** 'Q' Key
        db '1'       ;Matrix Data Code 16, ******** '1' Key
        db 0FFH      ;Matrix Data Code 17, Not Used
        db 0FFH      ;Matrix Data Code 18, Not Used
        db 0FFH      ;Matrix Data Code 19, Not Used
        db 'Z'       ;Matrix Data Code 1A, ******** 'Z' Key
        db 'S'       ;Matrix Data Code 1B, ******** 'S' Key
        db 'A'       ;Matrix Data Code 1C, ******** 'A' Key
        db 'W'       ;Matrix Data Code 1D, ******** 'W' Key
        db '2'       ;Matrix Data Code 1E, ******** '2' Key
        db 0FFH      ;Matrix Data Code 1F, Not Used

        db 0FFH      ;Matrix Data Code 20, Not Used
        db 'C'       ;Matrix Data Code 21, ******** 'C' Key
        db 'X'       ;Matrix Data Code 22, ******** 'X' Key
        db 'D'       ;Matrix Data Code 23, ******** 'D' Key
        db 'E'       ;Matrix Data Code 24, ******** 'E' Key
        db 0FFH        ;Matrix Data Code FC, Not Used
        db 0FFH        ;Matrix Data Code FD, Not Used
        db 0FFH        ;Matrix Data Code FE, Not Used
        db 0FFH        ;Matrix Data Code FF, Not Used


;The following Lookup Table is used to generate ASCII codes
;from certain Keys on the Keyboard, to display labels on
;the LCD.  While in the Label Mode, the user presses a Key
;on the Keyboard.  The code generated by the Key press
;is used in this lookup table to find the corresponding ASCII
;character for that Key.  In the Label Mode, The LCD only displays
;letters, numbers and some punctuation.  All letters are capitalized.
;Special Keys and Function Keys (other than backspace
;and enter) are ignored during the Label Mode.

Label_Lookup:
        db 0FFH        ;Matrix Data Code 0, Not Used
        db 0FFH        ;Matrix Data Code 1, Not Used
        db 0FFH        ;Matrix Data Code 2, Not Used
        db 0FFH        ;Matrix Data Code 3, Not Used
        db 0FFH        ;Matrix Data Code 4, Not Used
        db 0FFH        ;Matrix Data Code 5, Not Used
        db 0FFH        ;Matrix Data Code 6, Not Used
        db 0FFH        ;Matrix Data Code 7, Not Used
        db 0FFH        ;Matrix Data Code 8, Not Used
        db 0FFH        ;Matrix Data Code 9, Not Used
        db 0FFH        ;Matrix Data Code A, Not Used
        db 0FFH        ;Matrix Data Code B, Not Used
        db 0FFH        ;Matrix Data Code C, Not Used
        db 0FFH        ;Matrix Data Code D, Not Used
        db 0FFH        ;Matrix Data Code E, Not Used
        db 0FFH        ;Matrix Data Code F, Not Used

        db 0FFH        ;Matrix Data Code 10, Not Used
        db 0FFH        ;Matrix Data Code 11, Not Used
        db 0FFH        ;Matrix Data Code 12, Not Used
        db 0FFH        ;Matrix Data Code 13, Not Used
        db 0FFH        ;Matrix Data Code 14, Not Used
        db 'Q'         ;Matrix Data Code 15, ******** 'Q' Key
        db '1'         ;Matrix Data Code 16, ******** '1' Key
        db 0FFH        ;Matrix Data Code 17, Not Used
        db 0FFH        ;Matrix Data Code 18, Not Used
        db 0FFH        ;Matrix Data Code 19, Not Used
        db 'Z'         ;Matrix Data Code 1A, ******** 'Z' Key
        db 'S'         ;Matrix Data Code 1B, ******** 'S' Key
        db 'A'         ;Matrix Data Code 1C, ******** 'A' Key
        db 'W'         ;Matrix Data Code 1D, ******** 'W' Key
        db '2'         ;Matrix Data Code 1E, ******** '2' Key
        db 0FFH        ;Matrix Data Code 1F, Not Used

        db 0FFH        ;Matrix Data Code 20, Not Used
        db 'C'         ;Matrix Data Code 21, ******** 'C' Key
        db 'X'         ;Matrix Data Code 22, ******** 'X' Key
        db 'D'         ;Matrix Data Code 23, ******** 'D' Key
        db 'E'         ;Matrix Data Code 24, ******** 'E' Key
db '4'        ;Matrix Data Code 25, ******** '4' Key
db '3'        ;Matrix Data Code 26, ******** '3' Key
db 0FFH       ;Matrix Data Code 27, Not Used
db 0FFH       ;Matrix Data Code 28, Not Used
db ' '        ;Matrix Data Code 29, ******** ' ' Key
db 'V'        ;Matrix Data Code 2A, ******** 'V' Key
db 'F'        ;Matrix Data Code 2B, ******** 'F' Key
db 'T'        ;Matrix Data Code 2C, ******** 'T' Key
db 'R'        ;Matrix Data Code 2D, ******** 'R' Key
db '5'        ;Matrix Data Code 2E, ******** '5' Key
db 0FFH       ;Matrix Data Code 2F, Not Used

db 0FFH       ;Matrix Data Code 30, Not Used
db 'N'        ;Matrix Data Code 31, ******** 'N' Key
db 'B'        ;Matrix Data Code 32, ******** 'B' Key
db 'H'        ;Matrix Data Code 33, ******** 'H' Key
db 'G'        ;Matrix Data Code 34, ******** 'G' Key
db 'Y'        ;Matrix Data Code 35, ******** 'Y' Key
db '6'        ;Matrix Data Code 36, ******** '6' Key
db 0FFH       ;Matrix Data Code 37, Not Used
db 0FFH       ;Matrix Data Code 38, Not Used
db 0FFH       ;Matrix Data Code 39, Not Used
db 'M'        ;Matrix Data Code 3A, ******** 'M' Key
db 'J'        ;Matrix Data Code 3B, ******** 'J' Key
db 'U'        ;Matrix Data Code 3C, ******** 'U' Key
db '7'        ;Matrix Data Code 3D, ******** '7' Key
db '8'        ;Matrix Data Code 3E, ******** '8' Key
db 0FFH       ;Matrix Data Code 3F, Not Used

db 0FFH       ;Matrix Data Code 40, Not Used
db ','        ;Matrix Data Code 41, ******** ',' Key
db 'K'        ;Matrix Data Code 42, ******** 'K' Key
db 'I'        ;Matrix Data Code 43, ******** 'I' Key
db 'O'        ;Matrix Data Code 44, ******** 'O' Key
db '0'        ;Matrix Data Code 45, ******** '0' Key
db '9'        ;Matrix Data Code 46, ******** '9' Key
db 0FFH       ;Matrix Data Code 47, Not Used
db 0FFH       ;Matrix Data Code 48, Not Used
db '.'        ;Matrix Data Code 49, ******** '.' Key
db '/'        ;Matrix Data Code 4A, ******** '/' Key
db 'L'        ;Matrix Data Code 4B, ******** 'L' Key
db ';'        ;Matrix Data Code 4C, ******** ';' Key
db 'P'        ;Matrix Data Code 4D, ******** 'P' Key
db '-'        ;Matrix Data Code 4E, ******** '-' Key
db 0FFH       ;Matrix Data Code 4F, Not Used

db 0FFH       ;Matrix Data Code 50, Not Used
db 0FFH       ;Matrix Data Code 51, Not Used
db 027H       ;Matrix Data Code 52, ******** 'Single Quote' Key
db 0FFH       ;Matrix Data Code 53, Not Used
db '['        ;Matrix Data Code 54, ******** '[' Key
db '='        ;Matrix Data Code 55, ******** '=' Key
db 0FFH       ;Matrix Data Code 56, Not Used
db 0FFH       ;Matrix Data Code 57, Not Used
db 0FFH       ;Matrix Data Code 58, Not Used
db 0FFH       ;Matrix Data Code 59, Not Used
db 0FFH       ;Matrix Data Code 5A, Not Used
db ']'        ;Matrix Data Code 5B, ******** ']' Key
db 0FFH       ;Matrix Data Code 5C, Not Used
db '\'        ;Matrix Data Code 5D, ******** '\' Key
db 0FFH       ;Matrix Data Code 5E, Not Used
db 0FFH       ;Matrix Data Code 5F, Not Used

db 0FFH       ;Matrix Data Code 60, Not Used
db 0FFH       ;Matrix Data Code 61, Not Used
db 0FFH       ;Matrix Data Code 62, Not Used
db 0FFH       ;Matrix Data Code 63, Not Used
db 0FFH       ;Matrix Data Code 64, Not Used
db 0FFH       ;Matrix Data Code 65, Not Used
db 0FFH       ;Matrix Data Code 66, Not Used
db 0FFH       ;Matrix Data Code 67, Not Used
db 0FFH       ;Matrix Data Code 68, Not Used
db 0FFH       ;Matrix Data Code 69, Not Used
db 0FFH       ;Matrix Data Code 6A, Not Used
db 0FFH       ;Matrix Data Code 6B, Not Used
db 0FFH       ;Matrix Data Code 6C, Not Used
db 0FFH       ;Matrix Data Code 6D, Not Used
db 0FFH       ;Matrix Data Code 6E, Not Used
db 0FFH       ;Matrix Data Code 6F, Not Used

db 0FFH       ;Matrix Data Code 70, Not Used
db 0FFH       ;Matrix Data Code 71, Not Used
db 0FFH       ;Matrix Data Code 72, Not Used
db 0FFH       ;Matrix Data Code 73, Not Used
db 0FFH       ;Matrix Data Code 74, Not Used
db 0FFH       ;Matrix Data Code 75, Not Used
db 0FFH       ;Matrix Data Code 76, Not Used
db 0FFH       ;Matrix Data Code 77, Not Used
db 0FFH       ;Matrix Data Code 78, Not Used
db 0FFH       ;Matrix Data Code 79, Not Used
db 0FFH       ;Matrix Data Code 7A, Not Used
db 0FFH       ;Matrix Data Code 7B, Not Used
db 0FFH       ;Matrix Data Code 7C, Not Used
db 0FFH       ;Matrix Data Code 7D, Not Used
db 0FFH       ;Matrix Data Code 7E, Not Used
db 0FFH       ;Matrix Data Code 7F, Not Used

db 0FFH       ;Matrix Data Code 80, Not Used
db 0FFH       ;Matrix Data Code 81, Not Used
db 0FFH       ;Matrix Data Code 82, Not Used
db 0FFH       ;Matrix Data Code 83, Not Used
db 0FFH       ;Matrix Data Code 84, Not Used
db 0FFH       ;Matrix Data Code 85, Not Used
db 0FFH       ;Matrix Data Code 86, Not Used
db 0FFH       ;Matrix Data Code 87, Not Used
db 0FFH       ;Matrix Data Code 88, Not Used
db 0FFH       ;Matrix Data Code 89, Not Used
db 0FFH       ;Matrix Data Code 8A, Not Used
db 0FFH       ;Matrix Data Code 8B, Not Used
db 0FFH       ;Matrix Data Code 8C, Not Used
db 0FFH       ;Matrix Data Code 8D, Not Used
db 0FFH       ;Matrix Data Code 8E, Not Used
db 0FFH       ;Matrix Data Code 8F, Not Used
db 0FFH      ;Matrix Data Code 90, Not Used
db 0FFH      ;Matrix Data Code 91, Not Used
db 0FFH      ;Matrix Data Code 92, Not Used
db 0FFH      ;Matrix Data Code 93, Not Used
db 0FFH      ;Matrix Data Code 94, Not Used
db 0FFH      ;Matrix Data Code 95, Not Used
db 0FFH      ;Matrix Data Code 96, Not Used
db 0FFH      ;Matrix Data Code 97, Not Used
db 0FFH      ;Matrix Data Code 98, Not Used
db 0FFH      ;Matrix Data Code 99, Not Used
db 0FFH      ;Matrix Data Code 9A, Not Used
db 0FFH      ;Matrix Data Code 9B, Not Used
db 0FFH      ;Matrix Data Code 9C, Not Used
db 0FFH      ;Matrix Data Code 9D, Not Used
db 0FFH      ;Matrix Data Code 9E, Not Used
db 0FFH      ;Matrix Data Code 9F, Not Used

db 0FFH      ;Matrix Data Code A0, Not Used
db 0FFH      ;Matrix Data Code A1, Not Used
db 0FFH      ;Matrix Data Code A2, Not Used
db 0FFH      ;Matrix Data Code A3, Not Used
db 0FFH      ;Matrix Data Code A4, Not Used
db 0FFH      ;Matrix Data Code A5, Not Used
db 0FFH      ;Matrix Data Code A6, Not Used
db 0FFH      ;Matrix Data Code A7, Not Used
db 0FFH      ;Matrix Data Code A8, Not Used
db 0FFH      ;Matrix Data Code A9, Not Used
db 0FFH      ;Matrix Data Code AA, Not Used
db 0FFH      ;Matrix Data Code AB, Not Used
db 0FFH      ;Matrix Data Code AC, Not Used
db 0FFH      ;Matrix Data Code AD, Not Used
db 0FFH      ;Matrix Data Code AE, Not Used
db 0FFH      ;Matrix Data Code AF, Not Used

db 0FFH      ;Matrix Data Code B0, Not Used
db 0FFH      ;Matrix Data Code B1, Not Used
db 0FFH      ;Matrix Data Code B2, Not Used
db 0FFH      ;Matrix Data Code B3, Not Used
db 0FFH      ;Matrix Data Code B4, Not Used
db 0FFH      ;Matrix Data Code B5, Not Used
db 0FFH      ;Matrix Data Code B6, Not Used
db 0FFH      ;Matrix Data Code B7, Not Used
db 0FFH      ;Matrix Data Code B8, Not Used
db 0FFH      ;Matrix Data Code B9, Not Used
db 0FFH      ;Matrix Data Code BA, Not Used
db 0FFH      ;Matrix Data Code BB, Not Used
db 0FFH      ;Matrix Data Code BC, Not Used
db 0FFH      ;Matrix Data Code BD, Not Used
db 0FFH      ;Matrix Data Code BE, Not Used
db 0FFH      ;Matrix Data Code BF, Not Used

db 0FFH      ;Matrix Data Code C0, Not Used
db 0FFH      ;Matrix Data Code C1, Not Used
db 0FFH      ;Matrix Data Code C2, Not Used
db 0FFH      ;Matrix Data Code C3, Not Used
db 0FFH      ;Matrix Data Code C4, Not Used
db 0FFH      ;Matrix Data Code C5, Not Used
db 0FFH       ;Matrix Data Code C6, Not Used
db 0FFH       ;Matrix Data Code C7, Not Used
db 0FFH       ;Matrix Data Code C8, Not Used
db 0FFH       ;Matrix Data Code C9, Not Used
db 0FFH       ;Matrix Data Code CA, Not Used
db 0FFH       ;Matrix Data Code CB, Not Used
db 0FFH       ;Matrix Data Code CC, Not Used
db 0FFH       ;Matrix Data Code CD, Not Used
db 0FFH       ;Matrix Data Code CE, Not Used
db 0FFH       ;Matrix Data Code CF, Not Used

db 0FFH       ;Matrix Data Code D0, Not Used
db 0FFH       ;Matrix Data Code D1, Not Used
db 0FFH       ;Matrix Data Code D2, Not Used
db 0FFH       ;Matrix Data Code D3, Not Used
db 0FFH       ;Matrix Data Code D4, Not Used
db 0FFH       ;Matrix Data Code D5, Not Used
db 0FFH       ;Matrix Data Code D6, Not Used
db 0FFH       ;Matrix Data Code D7, Not Used
db 0FFH       ;Matrix Data Code D8, Not Used
db 0FFH       ;Matrix Data Code D9, Not Used
db 0FFH       ;Matrix Data Code DA, Not Used
db 0FFH       ;Matrix Data Code DB, Not Used
db 0FFH       ;Matrix Data Code DC, Not Used
db 0FFH       ;Matrix Data Code DD, Not Used
db 0FFH       ;Matrix Data Code DE, Not Used
db 0FFH       ;Matrix Data Code DF, Not Used

db 0FFH       ;Matrix Data Code E0, Not Used
db 0FFH       ;Matrix Data Code E1, Not Used
db 0FFH       ;Matrix Data Code E2, Not Used
db 0FFH       ;Matrix Data Code E3, Not Used
db 0FFH       ;Matrix Data Code E4, Not Used
db 0FFH       ;Matrix Data Code E5, Not Used
db 0FFH       ;Matrix Data Code E6, Not Used
db 0FFH       ;Matrix Data Code E7, Not Used
db 0FFH       ;Matrix Data Code E8, Not Used
db 0FFH       ;Matrix Data Code E9, Not Used
db 0FFH       ;Matrix Data Code EA, Not Used
db 0FFH       ;Matrix Data Code EB, Not Used
db 0FFH       ;Matrix Data Code EC, Not Used
db 0FFH       ;Matrix Data Code ED, Not Used
db 0FFH       ;Matrix Data Code EE, Not Used
db 0FFH       ;Matrix Data Code EF, Not Used

db 0FFH       ;Matrix Data Code F0, Not Used
db 0FFH       ;Matrix Data Code F1, Not Used
db 0FFH       ;Matrix Data Code F2, Not Used
db 0FFH       ;Matrix Data Code F3, Not Used
db 0FFH       ;Matrix Data Code F4, Not Used
db 0FFH       ;Matrix Data Code F5, Not Used
db 0FFH       ;Matrix Data Code F6, Not Used
db 0FFH       ;Matrix Data Code F7, Not Used
db 0FFH       ;Matrix Data Code F8, Not Used
db 0FFH       ;Matrix Data Code F9, Not Used
db 0FFH       ;Matrix Data Code FA, Not Used
db 0FFH       ;Matrix Data Code FB, Not Used
```

```
        db 0FFH     ;Matrix Data Code FC, Not Used
        db 0FFH     ;Matrix Data Code FD, Not Used
        db 0FFH     ;Matrix Data Code FE, Not Used
        db 0FFH     ;Matrix Data Code FF, Not Used
#ASM_END


Get_Keys:
        ;initialize Key label
        Key_Label$=""

        IF B_Key_Code=76H THEN Key_Label$=" Escape"
        IF B_Key_Code=05H THEN Key_Label$=" F1    "
        IF B_Key_Code=06H THEN Key_Label$=" F2    "
        IF B_Key_Code=04H THEN Key_Label$=" F3    "
        IF B_Key_Code=0CH THEN Key_Label$=" F4    "
        IF B_Key_Code=03H THEN Key_Label$=" F5    "
        IF B_Key_Code=0BH THEN Key_Label$=" F6    "
        IF B_Key_Code=83H THEN Key_Label$=" F7    "
        IF B_Key_Code=0AH THEN Key_Label$=" F8    "
        IF B_Key_Code=01H THEN Key_Label$=" F9    "
        IF B_Key_Code=09H THEN Key_Label$=" F10   "
        IF B_Key_Code=78H THEN Key_Label$=" F11   "
        IF B_Key_Code=07H THEN Key_Label$=" F12   "

        IF B_Key_Code=0EH THEN Key_Label$=" Tilda "
        IF B_Key_Code=16H THEN Key_Label$=" 1     "
        IF B_Key_Code=1EH THEN Key_Label$=" 2     "
        IF B_Key_Code=26H THEN Key_Label$=" 3     "
        IF B_Key_Code=25H THEN Key_Label$=" 4     "
        IF B_Key_Code=2EH THEN Key_Label$=" 5     "
        IF B_Key_Code=36H THEN Key_Label$=" 6     "
        IF B_Key_Code=3DH THEN Key_Label$=" 7     "
        IF B_Key_Code=3EH THEN Key_Label$=" 8     "
        IF B_Key_Code=46H THEN Key_Label$=" 9     "
        IF B_Key_Code=45H THEN Key_Label$=" 0     "
        IF B_Key_Code=4EH THEN Key_Label$=" -     "
        IF B_Key_Code=55H THEN Key_Label$=" =     "
        IF B_Key_Code=5DH THEN Key_Label$=" \     "
        IF B_Key_Code=66H THEN Key_Label$=" BkSpce"

        IF B_Key_Code=0DH THEN Key_Label$=" Tab   "
        IF B_Key_Code=15H THEN Key_Label$=" Q     "
        IF B_Key_Code=1DH THEN Key_Label$=" W     "
        IF B_Key_Code=24H THEN Key_Label$=" E     "
        IF B_Key_Code=2DH THEN Key_Label$=" R     "
        IF B_Key_Code=2CH THEN Key_Label$=" T     "
        IF B_Key_Code=35H THEN Key_Label$=" Y     "
        IF B_Key_Code=3CH THEN Key_Label$=" U     "
        IF B_Key_Code=43H THEN Key_Label$=" I     "
        IF B_Key_Code=44H THEN Key_Label$=" O     "
        IF B_Key_Code=4DH THEN Key_Label$=" P     "
        IF B_Key_Code=54H THEN Key_Label$=" [     "
        IF B_Key_Code=5BH THEN Key_Label$=" ]     "
        IF B_Key_Code=5AH THEN Key_Label$=" Return"

        IF B_Key_Code=58H THEN Key_Label$=" Caps  "
        IF B_Key_Code=1CH THEN Key_Label$=" A      "
        IF B_Key_Code=1BH THEN Key_Label$=" S      "
        IF B_Key_Code=23H THEN Key_Label$=" D      "
        IF B_Key_Code=2BH THEN Key_Label$=" F      "
        IF B_Key_Code=34H THEN Key_Label$=" G      "
        IF B_Key_Code=33H THEN Key_Label$=" H      "
        IF B_Key_Code=3BH THEN Key_Label$=" J      "
        IF B_Key_Code=42H THEN Key_Label$=" K      "
        IF B_Key_Code=4BH THEN Key_Label$=" L      "
        IF B_Key_Code=4CH THEN Key_Label$=" ;      "
        IF B_Key_Code=52H THEN Key_Label$=" '      "

        IF B_Key_Code=12H THEN Key_Label$=" LShift"
        IF B_Key_Code=1AH THEN Key_Label$=" Z      "
        IF B_Key_Code=22H THEN Key_Label$=" X      "
        IF B_Key_Code=21H THEN Key_Label$=" C      "
        IF B_Key_Code=2AH THEN Key_Label$=" V      "
        IF B_Key_Code=32H THEN Key_Label$=" B      "
        IF B_Key_Code=31H THEN Key_Label$=" N      "
        IF B_Key_Code=3AH THEN Key_Label$=" M      "
        IF B_Key_Code=41H THEN Key_Label$=" ,      "
        IF B_Key_Code=49H THEN Key_Label$=" .      "
        IF B_Key_Code=4AH THEN Key_Label$=" /      "
        IF B_Key_Code=59H THEN Key_Label$=" RShift"

        IF B_Key_Code=14H THEN Key_Label$=" L Ctrl"
        IF B_Key_Code=11H THEN Key_Label$=" L Alt "
        IF B_Key_Code=29H THEN Key_Label$=" Space "

        IF B_Key_Code=7EH THEN Key_Label$=" Scroll"

        IF B_Key_Code=77H THEN Key_Label$=" Num    "
        IF B_Key_Code=7CH THEN Key_Label$=" Kpd *  "
        IF B_Key_Code=7BH THEN Key_Label$=" Kpd -  "

        IF B_Key_Code=6CH THEN Key_Label$=" Kpd 7  "
        IF B_Key_Code=75H THEN Key_Label$=" Kpd 8  "
        IF B_Key_Code=7DH THEN Key_Label$=" Kpd 9  "
        IF B_Key_Code=79H THEN Key_Label$=" Kpd +  "

        IF B_Key_Code=6BH THEN Key_Label$=" Kpd 4  "
        IF B_Key_Code=73H THEN Key_Label$=" Kpd 5  "
        IF B_Key_Code=74H THEN Key_Label$=" Kpd 6  "

        IF B_Key_Code=69H THEN Key_Label$=" Kpd 1  "
        IF B_Key_Code=72H THEN Key_Label$=" Kpd 2  "
        IF B_Key_Code=7AH THEN Key_Label$=" Kpd 3  "

        IF B_Key_Code=70H THEN Key_Label$=" Kpd 0  "
        IF B_Key_Code=71H THEN Key_Label$=" Kpd .  "

        IF Key_Label$="" THEN Key_Label$=" ??     "
    Return


    Get_E0_Keys:
        ;initialize Key label
        Key_Label$=""

        IF B_Key-Code=70H THEN Key_Label$=" Insert"
        IF B_Key_Code=6CH THEN Key_Label$=" Home  "
        IF B_Key_Code=7DH THEN Key_Label$=" Pg Up "

        IF B_Key_Code=71H THEN Key_Label$=" Delete"
        IF B_Key_Code=69H THEN Key_Label$=" End   "
        IF B_Key_Code=7AH THEN Key_Label$=" PgDown"

        IF B_Key_Code=75H THEN Key_Label$=" Up    "
        IF B_Key_Code=6BH THEN Key_Label$=" Left  "
        IF B_Key_Code=72H THEN Key_Label$=" Down  "
        IF B_Key_Code=74H THEN Key_Label$=" Right "

        IF B_Key_Code=4AH THEN Key_Label$=" Kpd / "
        IF B_Key_Code=5AH THEN Key_Label$=" KpdRet"

        IF B_Key_Code=11H THEN Key_Label$=" R Alt "
        IF B_Key_Code=14H THEN Key_Label$=" R Ctrl"

        IF Key_Label$="" THEN Key_Label$=" ??    "
Return

Get_Matrix_Pos:
        ;initialize Matrix position label
        Matrix_Position$=""

        IF B_Matrix_Code=46H THEN Matrix_Position$="A1"
        IF B_Matrix_Code=4EH THEN Matrix_Position$="B1"
        IF B_Matrix_Code=4CH THEN Matrix_Position$="C1"
        IF B_Matrix_Code=58H THEN Matrix_Position$="D1"
        IF B_Matrix_Code=04H THEN Matrix_Position$="E1"
        IF B_Matrix_Code=43H THEN Matrix_Position$="F1"
        IF B_Matrix_Code=3CH THEN Matrix_Position$="G1"
        IF B_Matrix_Code=3AH THEN Matrix_Position$="H1"
        IF B_Matrix_Code=7CH THEN Matrix_Position$="I1"
        IF B_Matrix_Code=2DH THEN Matrix_Position$="J1"

        IF B_Matrix_Code=76H THEN Matrix_Position$="A2"
        IF B_Matrix_Code=2BH THEN Matrix_Position$="B2"
        IF B_Matrix_Code=05H THEN Matrix_Position$="C2"
        IF B_Matrix_Code=09H THEN Matrix_Position$="D2"
        IF B_Matrix_Code=72H THEN Matrix_Position$="E2"
        IF B_Matrix_Code=7AH THEN Matrix_Position$="F2"
        IF B_Matrix_Code=6BH THEN Matrix_Position$="G2"
        IF B_Matrix_Code=73H THEN Matrix_Position$="H2"
        IF B_Matrix_Code=22H THEN Matrix_Position$="I2"
        IF B_Matrix_Code=35H THEN Matrix_Position$="J2"

        IF B_Matrix_Code=1AH THEN Matrix_Position$="A3"
        IF B_Matrix_Code=23H THEN Matrix_Position$="B3"
        IF B_Matrix_Code=24H THEN Matrix_Position$="C3"
        IF B_Matrix_Code=5AH THEN Matrix_Position$="D3"
        IF B_Matrix_Code=71H THEN Matrix_Position$="E3"
        IF B_Matrix_Code=79H THEN Matrix_Position$="F3"
        IF B_Matrix_Code=70H THEN Matrix_Position$="G3"
IF B_Matrix_Code=69H THEN Matrix_Position$="H3"
IF B_Matrix_Code=2AH THEN Matrix_Position$="I3"
IF B_Matrix_Code=1DH THEN Matrix_Position$="J3"

IF B_Matrix_Code=55H THEN Matrix_Position$="A4"
IF B_Matrix_Code=1CH THEN Matrix_Position$="B4"
IF B_Matrix_Code=32H THEN Matrix_Position$="C4"
IF B_Matrix_Code=66H THEN Matrix_Position$="D4"
IF B_Matrix_Code=31H THEN Matrix_Position$="E4"
IF B_Matrix_Code=44H THEN Matrix_Position$="F4"
IF B_Matrix_Code=4DH THEN Matrix_Position$="G4"
IF B_Matrix_Code=7BH THEN Matrix_Position$="H4"
IF B_Matrix_Code=2CH THEN Matrix_Position$="I4"
IF B_Matrix_Code=0DH THEN Matrix_Position$="J4"

IF B_Matrix_Code=4AH THEN Matrix_Position$="A5"
IF B_Matrix_Code=54H THEN Matrix_Position$="B5"
IF B_Matrix_Code=5BH THEN Matrix_Position$="C5"
IF B_Matrix_Code=0EH THEN Matrix_Position$="D5"
IF B_Matrix_Code=42H THEN Matrix_Position$="E5"
IF B_Matrix_Code=4BH THEN Matrix_Position$="F5"
IF B_Matrix_Code=11H THEN Matrix_Position$="G5"
IF B_Matrix_Code=14H THEN Matrix_Position$="H5"
IF B_Matrix_Code=7EH THEN Matrix_Position$="I5"
IF B_Matrix_Code=29H THEN Matrix_Position$="J5"

IF B_Matrix_Code=3EH THEN Matrix_Position$="A6"
IF B_Matrix_Code=52H THEN Matrix_Position$="B6"
IF B_Matrix_Code=5DH THEN Matrix_Position$="C6"
IF B_Matrix_Code=49H THEN Matrix_Position$="D6"
IF B_Matrix_Code=77H THEN Matrix_Position$="E6"
IF B_Matrix_Code=34H THEN Matrix_Position$="F6"
IF B_Matrix_Code=33H THEN Matrix_Position$="G6"
IF B_Matrix_Code=3BH THEN Matrix_Position$="H6"
IF B_Matrix_Code=15H THEN Matrix_Position$="I6"
IF B_Matrix_Code=1BH THEN Matrix_Position$="J6"

IF B_Matrix_Code=25H THEN Matrix_Position$="A7"
IF B_Matrix_Code=2EH THEN Matrix_Position$="B7"
IF B_Matrix_Code=36H THEN Matrix_Position$="C7"
IF B_Matrix_Code=3DH THEN Matrix_Position$="D7"
IF B_Matrix_Code=03H THEN Matrix_Position$="E7"
IF B_Matrix_Code=0BH THEN Matrix_Position$="F7"
IF B_Matrix_Code=83H THEN Matrix_Position$="G7"
IF B_Matrix_Code=0AH THEN Matrix_Position$="H7"
IF B_Matrix_Code=75H THEN Matrix_Position$="I7"
IF B_Matrix_Code=7DH THEN Matrix_Position$="J7"

IF B_Matrix_Code=45H THEN Matrix_Position$="A8"
IF B_Matrix_Code=16H THEN Matrix_Position$="B8"
IF B_Matrix_Code=1EH THEN Matrix_Position$="C8"
IF B_Matrix_Code=26H THEN Matrix_Position$="D8"
IF B_Matrix_Code=78H THEN Matrix_Position$="E8"
IF B_Matrix_Code=07H THEN Matrix_Position$="F8"
IF B_Matrix_Code=06H THEN Matrix_Position$="G8"
IF B_Matrix_Code=0CH THEN Matrix_Position$="H8"
IF B_Matrix_Code=74H THEN Matrix_Position$="I8"
```

```
      IF B_Matrix_Code=6CH THEN Matrix_Position$="J8"

      IF Matrix_Position$="" THEN Matrix_Position$="??"
Return
```

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 September 2001 (20.09.2001)

PCT

(10) International Publication Number
WO 01/69366 A3

(51) International Patent Classification⁷: G06F 3/023, 3/02

(21) International Application Number: PCT/US01/07152

(22) International Filing Date: 7 March 2001 (07.03.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/524,011 13 March 2000 (13.03.2000) US

(71) Applicant: ERGODEX [US/US]; 1060 La Avenida, Mountain View, CA 94043-1422 (US).

(72) Inventor: RIX, Scott, M.; 505 Cypress Point Drive #104, Moutain View, CA 94043 (US).

(74) Agents: FRANKLIN, Eric, J. et al.; Swidler Berlin Shereff Friedman, LLP, Suite 300, 3000 K Street, NW, Washington, DC 20007 (US).

(81) Designated State *(national)*: JP.

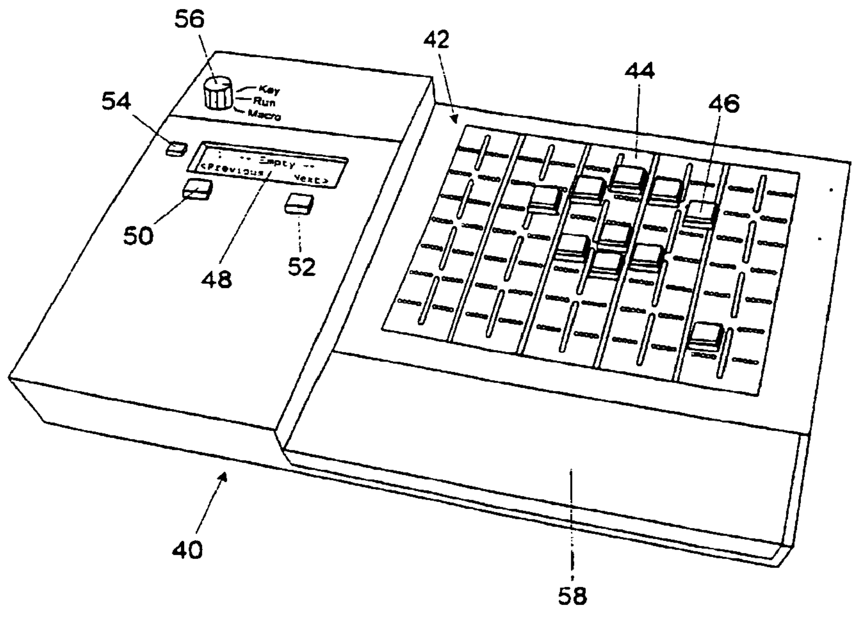
(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(88) Date of publication of the international search report: 13 June 2002

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: COMPUTER INPUT DEVICE WITH INDIVIDUALLY POSITIONABLE AND PROGRAMMABLE SWITCHES



(57) Abstract: A configurable computer input device. At least one switch is removably attachable to a surface and is in communication with a processor. At least one function is assignable to acitvation of the switch. The at least one switch may be repositioned distances smaller than a length or width of the at least one switch. Circuitry is in communication with the at least one switch for assigning at least one function to activation of the switch. Circuitry communicates the at least one function to a host computer. Circuitry determines the actuation status of the at least one switch and communicates the actuation status to the host computer.

WO 01/69366 A3

INTERNATIONAL SEARCH REPORT

Inte .onal Application No
PCT/US 01/07152

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F3/023 G06F3/02

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

WPI Data, PAJ, IBM-TDB, EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 4 906 117 A (BIRDWELL GERRY G) 6 March 1990 (1990-03-06)	1,2, 5-24, 26-28
	the whole document	
Y		3,4
A		25

X	"BOARDLESS TERMINAL KEYBOARD" IBM TECHNICAL DISCLOSURE BULLETIN, IBM CORP. NEW YORK, US, vol. 32, no. 10A, 1 March 1990 (1990-03-01), pages 82-84, XP000083171 ISSN: 0018-8689 the whole document	1,2,7, 10-17, 20-24, 26,28
Y		3,4

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search 15 April 2002	Date of mailing of the international search report 22/04/2002
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Davenport, K

2

INTERNATIONAL SEARCH REPORT

Inte. onal Application No
PCT/US 01/07152

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT		
Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 579 002 A (CRACK RICHARD ET AL) 26 November 1996 (1996-11-26) abstract column 2, line 20 -column 3, line 9 column 4, line 49 -column 6, line 36; figures 2,3 -----	1-28

2

INTERNATIONAL SEARCH REPORT

ıformation on patent family members

Inte onal Application No
PCT/US 01/07152

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 4906117	A	06-03-1990	JP	2558263 B2	27-11-1996
			JP	62160617 A	16-07-1987
US 5579002	A	26-11-1996	US	5729222 A	17-03-1998
			AU	6018494 A	20-12-1994
			WO	9428634 A1	08-12-1994